A Survey of Web Clustering Engines

CLAUDIO CARPINETO

Fondazione Ugo Bordoni

STANISIAW OSIŃSKI

Carrot Search

GIOVANNI ROMANO

Fondazione Ugo Bordoni

and

DAWID WEISS

Poznan University of Technology

Web clustering engines organize search results by topic, thus offering a complementary view to the flat-ranked list returned by conventional search engines. In this survey, we discuss the issues that must be addressed in the development of a Web clustering engine, including acquisition and preprocessing of search results, their clustering and visualization. Search results clustering, the core of the system, has specific requirements that cannot be addressed by classical clustering algorithms. We emphasize the role played by the quality of the cluster labels as opposed to optimizing only the clustering structure. We highlight the main characteristics of a number of existing Web clustering engines and also discuss how to evaluate their retrieval performance. Some directions for future research are finally presented.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*

General Terms: Algorithms, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Information retrieval, meta search engines, text clustring, search results clustering, user interfaces

Authors' addresses: C. Carpineto, Fondazione Ugo Bordoni, Via Baldassarre Castiglione 59, 00142 Roma, Italy; email: carpinet@fub.it; D. Weiss, Poznan University of Technology, ul. Piotrowo 2, 60-965 Poznan, Poland; email: dawid.weiss@cs.put.poznan.pl.

DOI 10.1145/1541880.1541884 http://doi.acm.org/10.1145/1541880.1541884

ACM Reference Format:
Carpineto, C., Osiński, S., Romano, G., and Weiss, D. 2009. A survey of web clustering engines. ACM Comput. Surv. 41, 3, Article 17 (July 2009), 38 pages.
DOI = 10.1145/1541880.1541884 http://doi.acm.org/10.1145/1541880.1541884

1. INTRODUCTION

Search engines are an invaluable tool for retrieving information from the Web. In response to a user query, they return a list of results ranked in order of relevance to the query. The user starts at the top of the list and follows it down examining one result at a time, until the sought information has been found.

However, while search engines are definitely good for certain search tasks such as finding the home page of an organization, they may be less effective for satisfying broad or ambiguous queries. The results on different subtopics or meanings of a query will be mixed together in the list, thus implying that the user may have to sift through a large number of irrelevant items to locate those of interest. On the other hand, there is no way to exactly figure out what is relevant to the user given that the queries are usually very short and their interpretation is inherently ambiguous in the absence of a context.

A different approach to Web information retrieval is based on categorizing the whole Web, whether manually or automatically, and then letting the user see the results associated with the categories that best match his or her query. However, even the largest Web directories such as the Open Directory Project[1] cover only a small fraction of the existing pages. Furthermore, the directory may be of little help for a particular user query, or for a particular aspect of it. In fact, the Web directories are most often used to influence the output of a direct search in response to common user queries.

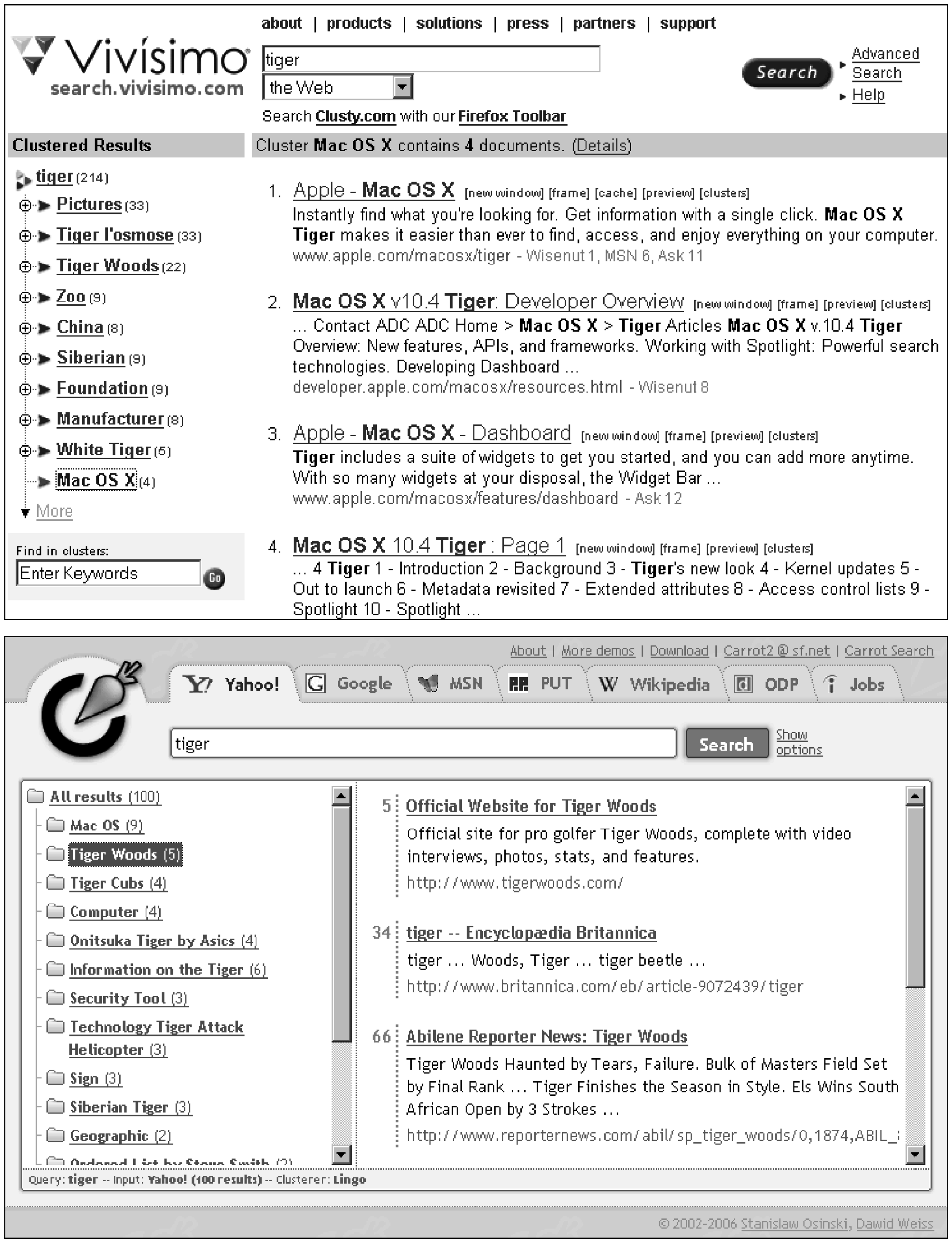
A third method is search results clustering, which consists of grouping the results returned by a search engine into a hierarchy of labeled clusters (also called categories). This method combines the best features of query-based and category-based search, in that the user may focus on a general topic by a weakly-specified query, and then drill down through the highly-specific themes that have been dynamically created from the query results. The main advantage of the cluster hierarchy is that it makes for shortcuts to the items that relate to the same meaning. In addition, it allows better topic understanding and favors systematic exploration of search results.

To illustrate, Figure 1 shows the clustered results returned for the query "tiger" (as of December 2006). Like many queries on the Web, "tiger" has multiple meanings with several high-scoring hits each: the feline, the Mac OS X computer operating system, the golf champion, the UNIX security tool, the Chinese horoscope, the mapping service, and so on. These different meanings are well represented in Figure 1.

By contrast, if we submit the query "tiger" to Google or Yahoo!, we see that each meaning's items are scattered in the ranked list of search results, often through a large number of result pages. Major search engines recognize this problem and interweave documents related to different topics—a technique known as *implicit clustering*—but given the limited capacity of a single results page, certain topics will be inevitably hidden from the user. On the same query, the Open Directory does a good job in grouping the items about the feline and the golf champion, but it completely fails to cover the other meanings.

The systems that perform clustering of Web search results, also known as clustering engines, have become rather popular in recent years. The first commercial clustering engine was probably Northern Light, at the end of the 1990s. It was based on a predefined

[1]www.dmoz.org


about | products | solutions | press | partners | support
Vivísimo
search.vivisimo.com
tiger
the Web
Search
Advanced Search
Help
Search Clusty.com with our Firefox Toolbar
Find in clusters:
Enter Keywords
Go
About | More demos | Download | Carrot2 @ sf.net | Carrot Search
Yahoo!
Google
MSN
PUT
Wikipedia
ODP
Jobs
tiger
Search
Show options
© 2002-2006 Stanisław Osiński, Dawid Weiss

Clustered Results

- tiger (214)
- Pictures (33)
- Tiger l'osmose (33)
- Tiger Woods (22)
- Zoo (9)
- China (8)
- Siberian (9)
- Foundation (9)
- Manufacturer (8)
- White Tiger (5)
- Mac OS X (4)
- More

Cluster **Mac OS X** contains **4** documents. (Details)

1. Apple - **Mac OS X** [new window] [frame] [cache] [preview] [clusters]
Instantly find what you're looking for. Get information with a single click. **Mac OS X Tiger** makes it easier than ever to find, access, and enjoy everything on your computer.
www.apple.com/macosx/tiger - Wisenut 1, MSN 6, Ask 11

2. **Mac OS X** v10.4 **Tiger**: Developer Overview [new window] [frame] [preview] [clusters]
... Contact ADC ADC Home > **Mac OS X** > **Tiger** Articles **Mac OS X** v.10.4 **Tiger** Overview: New features, APIs, and frameworks. Working with Spotlight: Powerful search technologies. Developing Dashboard ...
developer.apple.com/macosx/resources.html - Wisenut 8

3. Apple - **Mac OS X** - Dashboard [new window] [frame] [preview] [clusters]
Tiger includes a suite of widgets to get you started, and you can add more anytime. With so many widgets at your disposal, the Widget Bar ...
www.apple.com/macosx/features/dashboard - Ask 12

4. **Mac OS X** 10.4 **Tiger** : Page 1 [new window] [frame] [preview] [clusters]
... 4 **Tiger** 1 - Introduction 2 - Background 3 - **Tiger**'s new look 4 - Kernel updates 5 - Out to launch 6 - Metadata revisited 7 - Extended attributes 8 - Access control lists 9 - Spotlight 10 - Spotlight ...

- All results (100)
  - Mac OS (9)
  - Tiger Woods (5)
  - Tiger Cubs (4)
  - Computer (4)
  - Onitsuka Tiger by Asics (4)
  - Information on the Tiger (6)
  - Security Tool (3)
  - Technology Tiger Attack Helicopter (3)
  - Sign (3)
  - Siberian Tiger (3)
  - Geographic (2)
  - Ordered List by Steve Smith (2)

5 Official Website for Tiger Woods
Official site for pro golfer Tiger Woods, complete with video interviews, photos, stats, and features.
http://www.tigerwoods.com/

34 tiger -- Encyclopædia Britannica
tiger ... Woods, Tiger ... tiger beetle ...
http://www.britannica.com/eb/article-9072439/tiger

66 Abilene Reporter News: Tiger Woods
Tiger Woods Haunted by Tears, Failure. Bulk of Masters Field Set by Final Rank ... Tiger Finishes the Season in Style. Els Wins South African Open by 3 Strokes ...
http://www.reporternews.com/abil/sp_tiger_woods/0,1874,ABIL_

Query: **tiger** -- Input: **Yahoo! (100 results)** -- Clusterer: **Lingo**

Fig. 1. Clustered search results for the query "tiger." Screenshots from the commercial engine Vivísimo (top, www.vivivimo.com), with selection of cluster "Mac OS X," and the open source Carrot^2 (bottom, www.carrot2.org), with selection of cluster "Tiger Woods."

set of categories, to which the search results were assigned. A major breakthrough was then made by Vivísimo, whose clusters and cluster labels were dynamically generated from the search results. Vivísimo won the "best meta-search engine award" assigned by SearchEngineWatch.com from 2001 to 2003.

Nowadays, several commercial systems for topic clustering are available that may output different clusters and cluster labels for the same query, although little is known about the particular methods being employed. One common feature of most current clustering engines is that they do not maintain their own index of documents; similar to meta search engines [Meng et al. 2002], they take the search results from one or more publicly accessible search engines. Even the major search engines are becoming more involved in the clustering issue. Clustering by site (a form of clustering that groups the results that are on the same Web site) is currently provided by several commercial systems including Google. The concept (or entity) recognition that multiple search engines are working on (for finding e.g., people, products, books) also seems related to clustering of search results [Ye et al. 2003].

Not only has search results clustering attracted considerable commercial interest, but it is also an active research area, with a large number of published papers discussing specific issues and systems. Search results clustering is clearly related to the field of document clustering but it poses unique challenges concerning both the effectiveness and the efficiency of the underlying algorithms that cannot be addressed by conventional techniques. The main difference is the emphasis on the quality of cluster labels, whereas this issue was of somewhat lesser importance in earlier research on document clustering. The consequence has been the development of new algorithms, surveyed in this article, that are primarily focused on generating expressive cluster labels rather than optimal clusters. Note, however, that this article is not another survey of general-purpose (albeit advanced) clustering algorithms. Our focus is on the systems that perform search results clustering, in which such new algorithms are usually encompassed. We discuss the issues that must be addressed to develop a Web clustering engine and the technical solutions that can be adopted. We consider the whole processing pipeline, from search result acquisition to visualization of clustered results. We also review existing systems and discuss how to assess their retrieval performance.

Although there have been earlier attempts to review search results clustering [Leuski and Croft 1996; Maarek et al. 2000; Ferragina and Gulli 2005; Husek et al. 2006], their scope is usually limited to the clustering techniques, without considering the issues involved in the realization of a complete system. Furthermore, earlier reviews are mostly outdated and/or largely incomplete. To our knowledge, this is the first systematic review of Web clustering engines that deals with issues, algorithms, implementation, systems, and evaluation.

This article is intended for a large audience. Besides summarizing current practice for researchers in information retrieval and Web technologies, it may be of value to academic and industry practitioners interested in implementation and assessment of systems, as well as to scientific professionals who may want a snapshot of an evolving technology.

The rest of the article is organized in the following way. Section 2 discusses the main limitations of search engines that are addressed by clustering engines. Section 3 introduces the definitional and pragmatic features of search results clustering. Section 4 analyzes the main stages in which a clustering engine can be broken down—namely, acquisition of search results, input preprocessing, construction of clusters, and visualization of clustered results. The algorithms used in the clustering stage are classified based on the relative importance of cluster formation and cluster labeling. Section 5 provides an overview of the main existing systems whose technical details have been disclosed. Section 6 deals with computational efficiency and system implementation.

Section 7 is devoted to retrieval performance evaluation. We discuss how to compare the retrieval effectiveness of a clustering engine to that of a plain search engine and how to compare different clustering engines, including an experimental study of the subtopic retrieval performance of several clustering algorithms. In Section 8 directions for future work are highlighted and, finally, in Section 9 we present our conclusions.

2. THE GOAL OF WEB CLUSTERING ENGINES

Plain search engines are usually quite effective for certain types of search tasks, such as navigational queries (where the user has a particular URL to find) and transactional queries (where the user is interested in some Web-mediated activity). However, they can fail in addressing informational queries (in which the user has an information need to satisfy), which account for the majority of Web searches (see Broder [2002] and Rose and Levinson [2004] for a taxonomy of Web queries). This is especially true for informational searches expressed by vague, broad or ambiguous queries.

A clustering engine tries to address the limitations of current search engines by providing clustered results as an added feature to their standard user interface. We emphasize that clustering engines are usually seen as complementary—rather than alternative—to search engines. In fact, in most clustering engines the categories created by the system are kept separated from the plain result list, and users are allowed to use the list in the first place. The view that clustering engines are primarily helpful when search engines fail is also supported by some recent experimental studies of Web searches [Käki 2005; Teevan et al. 2004].

The search aspects where clustering engines can be most useful in complementing the output of plain search engines are the following.

—*Fast subtopic retrieval*. If the documents that pertain to the same subtopic have been correctly placed within the same cluster and the user is able to choose the right path from the cluster label, such documents can be accessed in logarithmic rather than linear time.

—*Topic exploration*. A cluster hierarchy provides a high-level view of the whole query topic including terms for query reformulation, which is particularly useful for informational searches in unknown or dynamic domains.

—*Alleviating information overlook*. Web searchers typically view only the first result page, thus overlooking most information. As a clustering engine summarizes the content of many search results in one single view on the first result page, the user may review hundreds of potentially relevant results without the need to download and scroll to subsequent pages.

In the next section we focus on search results clustering—the core component of a Web clustering engine—discussing how its features compare to those of traditional document clustering.

3. OVERVIEW OF SEARCH RESULTS CLUSTERING

Clustering is a broad field that studies general methods for the grouping of unlabeled data, with applications in many domains. In general, clustering can be characterized as a process of discovering subsets of objects in the input (*clusters*, *groups*) in such a way that objects within a cluster are similar to each other and objects from different clusters are dissimilar from each other, usually according to some similarity measure. Clustering has been the subject of several books (e.g., Hartigan [1975]; Everitt et al. [2001]) and a huge number of research papers, from a variety of perspectives (see Jain et al. [1999] for a comprehensive review).

Within general clustering, document clustering is a more focused field, closely related to this article's topics. Document clustering was proposed mainly as a method of improving the effectiveness of document ranking following the hypothesis that closely associated documents will match the same requests [van Rijsbergen 1979]. The canonical approach has been to cluster the entire collection in advance, typically into a hierarchical tree structure, and then return the documents contained in those clusters that best match the user's query (possibly sorted by score). Document clustering systems vary in the metrics used for measuring the similarity between documents and clusters, in the algorithm for building the cluster structure, and in the strategy for ranking the clustered documents against a query (see Willet [1988] for an early review and Manning et al. [2008] for an updated one).

As the computational time complexity of most classical hierarchical clustering algorithms is $O(n^2)$, where n is the number of documents to be clustered, this approach may become too slow for large and dynamic collections such as the Web. Even though clustering might take advantage of other similar computationally intensive activities that are usually performed offline such as duplicate or near-duplicate detection, its cost would further add to the inherent complexity of index maintenance.

For cluster-based Web information retrieval it is more convenient to follow a two-stage approach, in which clustering is performed as a postprocessing step on the set of documents retrieved by an information retrieval system on a query. Postretrieval clustering is not only more efficient than preretrieval clustering, but it may also be more effective. The reason is that preretrieval clustering might be based on features that are frequent in the collection but irrelevant for the query at hand, whereas postretrieval makes use only of query-specific features.

There are two types of postretrieval clustering. The clustering system may rerank the results and offer a new list to the user. In this case the system usually returns the items contained in one or more optimal clusters [Tombros et al. 2002; Liu and Croft 2004]. Alternatively, the clustering system groups the ranked results and gives the user the ability to choose the groups of interest in an interactive manner [Allen et al. 1993; Hearst and Pedersen 1996]. A clustering engine follows the latter approach, and the expression *search results clustering* usually refers to browsing a clustered collection of search results returned by a conventional Web search engine. Search results clustering can thus be seen as a particular subfield of clustering concerned with the identification of thematic groups of items in Web search results. The input and output of a search results clustering algorithm can be characterized more precisely in the following way.

The *input* is a set of search results obtained in response to a user query, each described by a URL, a title, and a snippet (a short text summarizing the context in which the query words appear in the result page). Assuming that there exists a logical topic structure in the result set, the *output* is a set of labeled clusters representing it in the closest possible way and organized in a set of flat partitions, hierarchy or other graph structure.

The dynamic nature of the data together with the interactive use of clustered results pose new requirements and challenges to clustering technology, as detailed in the following list.

—*Meaningful labels*. Traditional algorithms use the cluster centroid as cluster representative, which is of little use for guiding the user to locate the sought items. Each cluster label should concisely indicate the contents of the cluster items, while being consistent with the meanings of the labels of more general and more specific clusters.

—*Computational efficiency*. Search results clustering is performed online, within an application that requires overall subsecond response times. The critical step is the

Table I. Search Results Clustering Versus Traditional Document Clustering

Clustering Type	Cluster Labels	Cluster Computation	Input Data	Cluster Number	Cluster Intersection	GUI
Search results clustering	Natural language	Online	Snippets	Variable	Overlapping	Yes
Document clustering	Centroid	Offline	Documents	Fixed	Disjoint	No

acquisition of search results, whereas the efficiency of the cluster construction algorithm is less important due to the low number of input results.

—*Short input data description*. Due to computational reasons, the data available to the clustering algorithm for each search result are usually limited to a URL, an optional title, and a short excerpt of the document's text (the snippet). This contrasts with using more coherent, multidimensional data descriptions such as whole Web pages.

—*Unknown number of clusters*. Many traditional methods require this number as an input. In search results clustering, however, both the number and the size of clusters cannot be predetermined because they vary with the query; furthermore, they may have to be inferred from a variable number of search results.

—*Overlapping clusters*. As the same result may often be assigned to multiple themes, it is preferable to cater for overlapping clusters. Also, a graph may be more flexible than a tree because the latter does not easily permit recovery from bad decisions while traversing the cluster structure. Using a graph, the results contained in one cluster can be reached through several paths, each fitting a particular user's need or paradigm.

—*Graphical user interface (GUI)*. A clustering engine allows interactive browsing through clustered Web search results for a large population of users. It can thus take advantage of Web-based graphical interfaces to convey more visual information about the clusters and their relationships, provided that they are intuitive to use and computationally efficient.

The main differences between search results clustering and traditional document clustering are summarized in Table I.

In the next section we discuss how to address the conflicting issues of high accuracy and severe computational constraints in the development of a full Web clustering engine.

4. ARCHITECTURE AND TECHNIQUES OF WEB CLUSTERING ENGINES

Practical implementations of Web search clustering engines will usually consist of four general components: search results acquisition, input preprocessing, cluster construction, and visualization of clustered results, all arranged in a processing pipeline shown in Figure 2.

4.1. Search Results Acquisition

The task of the search results acquisition component is to provide input for the rest of the system. Based on the user-specified query string, the acquisition component must deliver typically between 50 and 500 search results, each of which should contain a title, a contextual snippet, and the URL pointing to the full text document being referred to.

A potential source of search results for the acquisition component is a public Web search engine, such as Yahoo!, Google, or Live Search. The most elegant way of

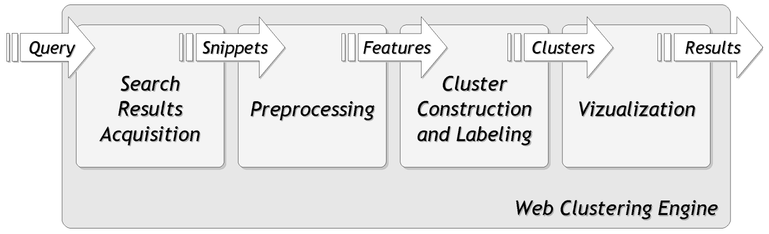


Fig. 2. Components of a Web search clustering engine.

Table II. Programmatic Interfaces to the Most Popular Search Engines and their Limitations as of December, 2007

Search Engine	Protocol	Queries Per Day	Results Per Search	Terms of Service
Alexa	SOAP or REST	n/a	20	Paid service (per-query).
Gigablast	REST/XML	100	10	Noncommercial use only.
Google	SOAP	1 000	10	Unsupported as of December 5, 2006. Noncommercial use only.
Google CSE	REST/XML	n/a	20	Custom search over selected sites/ domains. Paid service if XML feed is required.
MSN Search	SOAP	10 000	50	Per application-ID query limit. Noncommercial use only.
Yahoo!	REST/XML	5 000	100	Per-IP query limit. No commercial restrictions (except Local Search).

fetching results from such search engines is by using application programming interfaces (APIs) these engines provide. In the case of Yahoo!, for example, search results can be retrieved in a convenient XML format using a plain HTTP request (developer.yahoo.com/search/web/). This technique is commonly referred to as REST (Representational State Transfer), but the spectrum of technologies used to expose a search engine's resources varies depending on the vendor (see Table II).

At the time of writing, all major search engines (Google, Yahoo!, Live Search) provide public APIs, but they come with certain restrictions and limitations. These restrictions are technical (maximum number of queries per day, maximum number of fetched results per query), but also legal—terms of service allowing only non-commercial or research use. While the former can affect the overall performance of a clustering engine (we discuss this later in Section 6.1.1), the latter involves a risk of litigation. Access to search services is usually made available commercially for a fee (like in the business edition of Google's Custom Search Engine, www.google.com/coop/cse).

Another way of obtaining results from a public search engine is called *HTML scraping*. The search results acquisition component can use regular expressions or other form of markup detection to extract titles, snippets, and URLs from the HTML stream served by the search engine to its end users. Due to relatively frequent changes of the HTML markup, this method, if done manually, would be rather tedious and lead to reliability and maintenance problems. HTML scrapers can be trained using machine learning methods to extract the content automatically, but such techniques are not widespread [Zhao et al. 2005]. Additionally, even though this technique was popular in the early days of Web search clustering engines, it is nowadays strongly discouraged as it infringes search engines' legal terms of use (automated querying and processing of search results is typically prohibited). Besides, the public APIs we mentioned provide a more reliable and faster alternative.

There exists an encouraging movement to standardize access to search engines called Open Search (www.opensearch.org). A number of search feeds from various sources are already available, but unfortunately the ones from major vendors return results in HTML format only (suitable for scraping, but not for automated processing).

An alternative source of search results for the acquisition component is a dedicated document index. This scenario is particularly useful when there is a need to search and cluster documents that are not available through public search engines, for example, enterprise, domain-specific content or IR test collections. In this case, the acquisition component may additionally need to take responsibility for generating contextual document snippets.

4.2. Preprocessing of Search Results

Input preprocessing is a step that is common to all search results clustering systems. Its primary aim is to convert the contents of search results (output by the acquisition component) into a sequence of *features* used by the actual clustering algorithm. A typical cascade goes through language identification, tokenization, shallow language preprocessing (usually limited to stemming) and finally selection of features. We will take a closer look at each step.

Clustering engines that support multilingual content must perform initial *language recognition* on each search result in the input. Information about the language of each entry in the search result is required to choose a corresponding variant of subsequent components—tokenization and stemming algorithms—but it also helps during the feature selection phase by providing clues about common, unimportant words for each language (stop words) that can be ignored. Another challenge for language identification in search results clustering is the small length of input snippets provided for clustering. The trigram-based method, analyzed in Grefenstette [1995], was shown to be able to deal with this kind of input fairly successfully.

During the *tokenization* step, the text of each search result gets split into a sequence of basic independent units called *tokens*, which will usually represent single words, numbers, symbols and so on. Writing such a tokenizer for Latin languages would typically mean looking at spaces separating words and considering everything in between to be a single token [Manning and Schütze 1999]. This simple heuristic works well for the majority of the input, but fails to discover certain semantic tokens comprised of many words (*Big Apple* or *General Motors*, for example). Tokenization becomes much more complex for languages where white spaces are not present (such as Chinese) or where the text may switch direction (such as an Arabic text, within which English phrases are quoted).

An important quality of the tokenizer in search results clustering is noise-tolerance. The contextual document snippets provided as input are very likely to contain: ellipsis characters ("...") demarcating fragments of the full document, URLs, file names, or other characters having no easily interpretable meaning. A robust tokenizer should be able to identify and remove such noise and prepare a sequence of tokens suitable for shallow language preprocessing and feature extraction.

Finally, when binding a Web clustering engine to a source of search results that can expose documents as sequences of tokens rather than raw text, the tokenization step can be omitted in the clustering algorithms' processing pipeline. Section 6.2.3 provides more implementation-level details on such a setting.

Stemming is a typical shallow NLP technique. The aim of stemming is to remove the inflectional prefixes and suffixes of each word and thus reduce different grammatical forms of the word to a common base form called a *stem*. For example, the words *connected*, *connecting*, and *interconnection* would be transformed to the word *connect*. Note that while in this example all words transform to a single stem, which is also a dictionary word, this may not always be the case—a stem may not be a correct word. In fact it may not be a word at all—a stem is simply a unique token representing a certain set of words with roughly equal meaning. Because of this departure from real linguistics, stemming is considered a heuristic method and the entire process is dubbed *shallow* linguistic preprocessing (in contrast to finding true word lemmas). The most commonly used stemming algorithm for English is the Porter stemmer [Porter 1997]. Algorithms for several Latin-family languages can also be found in the Internet, for instance in the Snowball project.[2]

It should be said that researchers have been arguing [Kantrowitz et al. 2000] as to if and how stemming affects the quality of information retrieval systems, including clustering algorithms. When a great deal of input text is available, stemming does not seem to help much. On the other hand, it plays a crucial role when dealing with very limited content, such as search results, written in highly inflectional languages (as shown in Stefanowski and Weiss [2003a], where authors experiment with clustering search results in Polish).

Last but not least, the preprocessing step needs to extract features for each search result present in the input. In general data mining terms, features are atomic entities by which we can describe an object and represent its most important characteristic to an algorithm. When looking at text, the most intuitive set of features would be simply words of a given language, but this is not the only possibility. As shown in Table III, features used by different search results clustering systems vary from single words and fixed-length tuples of words (n-grams) to frequent phrases (variable-length sequences of words), and very algorithm-specific data structures, such as approximate sentences in the SnakeT system. A feature class with numerous possible values (like all words in a language) is often impractical since certain elements are closely related to each other and can form equivalent classes (for example all words with a unique stem), and other elements occur very infrequently or not at all. Many features are also irrelevant for the task of assessing similarity or dissimilarity between documents, and can be discarded. This is where feature extraction, selection and construction takes place. These techniques are well covered by a number of books and surveys (Yang and Pedersen [1997] and Liu et al. [2003] place particular emphasis on text processing), and we will omit their detailed description here, stopping at the conclusion that the choice of words, albeit the most common in text processing, does not exhaust the possibilities.

Regardless of what is extracted, the preprocessing phase ends with some information required to build the *model* of text representation—a model that is suitable for the task of document clustering and indirectly affects a difficult step that takes place later on—cluster labeling.

4.3. Cluster Construction and Labeling

The set of search results along with their features, extracted in the preprocessing step, are given as input to the clustering algorithm, which is responsible for building the clusters and labeling them. Because a large variety of search results clustering algorithms have been proposed, this raises the question of their classification. Clustering algorithms are usually classified according to the characteristics of their output structure, but here we take a different viewpoint.

[2]www.snowball.tartarus.org

In search results clustering, users are the ultimate consumers of clusters. A cluster which is labeled awkwardly, ambiguously, or nonsensically is very likely to be entirely omitted even if it points to a group of strongly related and somehow relevant documents. This observation led to a very important conclusion (credited to Vivísimo): in search results clustering description comes first. We would like to embrace this criterion and divide our discussion of algorithms according to how well they are prepared to produce sensible, comprehensive, and compact cluster labels. We distinguish three categories of algorithms: *data-centric*, *description-aware*, and *description-centric*. In the following sections we characterize challenges present in each group and provide a short description of a representative algorithm for reference.

4.3.1. Data-Centric Algorithms. A typical representative of this group consists of a conventional data clustering algorithm (hierarchical, optimization, spectral, or other), applied to the problem of clustering search results, often slightly modified to produce some kind of textual representation of the discovered clusters for end-users.

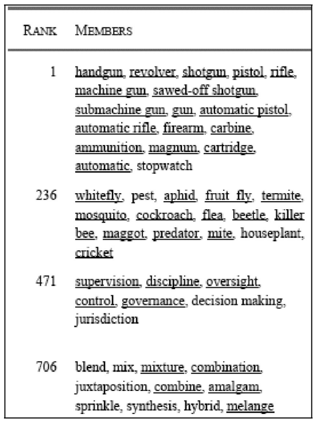
Scatter/Gather [Cutting et al. 1992; Hearst and Pedersen 1996] is a landmark example of a data-centric system. Developed in 1992 at Xerox PARC, Scatter/Gather is commonly perceived as a predecessor and conceptual parent of all postretrieval clustering systems that appeared later. The system worked by performing an initial clustering of a collection of documents into a set of k clusters (a process called *scattering*). This step could be performed offline and provided an initial overview of the collection of documents. Then, at querytime, the user selected clusters of interest and the system reclustered the indicated subcollection of documents dynamically (the *gathering* step). The query-recluster cycle is obviously very similar to search results clustering systems, with the difference that instead of querying a back-end search engine, Scatter/Gather *slices* its own collection of documents.

The data-centrism of Scatter/Gather becomes apparent by looking at its inner working. Let us briefly recall the concept of a vector space model (VSM) for text representation [Salton et al. 1975]. A document d is represented in the VSM as a vector $[w_{t_0}, w_{t_1}, \dots w_{t_\Omega}]$, where $t_0, t_1, \dots t_\Omega$ is a global set of words (features) and w_{t_i} expresses the weight (importance) of feature t_i to document d. Weights in a document vector typically reflect the distribution of occurrences of features in that document, possibly adjusted with a correcting factor that highlights the importance of features specific to that particular document in the context of the entire collection. For example, a term vector for the phrase "Polly had a dog and the dog had Polly" could appear as shown below (weights are simply counts of words, articles are rarely specific to any document and normally would be omitted).

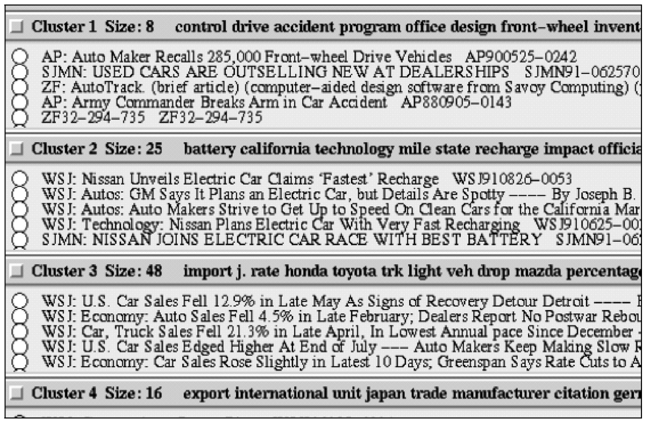
	a	and	dog	had	Polly	the
$d \rightarrow$	1	1	2	2	2	1

Note that once a text is converted to a document vector we can hardly speak of the text's *meaning*, because the vector is basically a collection of unrelated terms. For this reason the VSM is sometimes called a *bag-of-words* model.

Returning to Scatter/Gather, the clustering technique used there is the very popular agglomerative hierarchical clustering (AHC), with an average-link merge criterion [Everitt et al. 2001]. This technique essentially works by putting each document in its own cluster and then iteratively merging the two closest clusters until the desired number of k clusters is reached. The authors provide two additional heuristics—Buckshot and Fractionation—to keep the processing time linear rather than quadratic with the size of the input collection. The final result is a set of k clusters, each one described by



RANK	MEMBERS
1	handgun, revolver, shotgun, pistol, rifle, machine gun, sawed-off shotgun, submachine gun, gun, automatic pistol, automatic rifle, firearm, carbine, ammunition, magnum, cartridge, automatic, stopwatch
236	whitefly, pest, aphid, fruit fly, termite, mosquito, cockroach, flea, beetle, killer bee, maggot, predator, mite, houseplant, cricket
471	supervision, discipline, oversight, control, governance, decision making, jurisdiction
706	blend, mix, mixture, combination, juxtaposition, combine, amalgam, sprinkle, synthesis, hybrid, melange



```
Cluster 1  Size: 8     control drive accident program office design front–wheel invent
  AP: Auto Maker Recalls 285,000 Front–wheel Drive Vehicles   AP900525–0242
  SJMN: USED CARS ARE OUTSELLING NEW AT DEALERSHIPS   SJMN91–062570
  ZF: AutoTrack. (brief article) (computer–aided design software from Savoy Computing) (
  AP: Army Commander Breaks Arm in Car Accident   AP880905–0143
  ZF32–294–735   ZF32–294–735
Cluster 2  Size: 25    battery california technology mile state recharge impact officia
  WSJ: Nissan Unveils Electric Car Claims 'Fastest' Recharge   WSJ910826–0053
  WSJ: Autos: GM Says It Plans an Electric Car, but Details Are Spotty –––– By Joseph B.
  WSJ: Autos: Auto Makers Strive to Get Up to Speed On Clean Cars for the California Mar
  WSJ: Technology: Nissan Plans Electric Car With Very Fast Recharging   WSJ910625–00
  SJMN: NISSAN JOINS ELECTRIC CAR RACE WITH BEST BATTERY   SJMN91–06
Cluster 3  Size: 48    import j. rate honda toyota trk light veh drop mazda percentag
  WSJ: U.S. Car Sales Fell 12.9% in Late May As Signs of Recovery Detour Detroit ––––
  WSJ: Economy: Auto Sales Fell 4.5% in Late February; Dealers Report No Postwar Rebou
  WSJ: Car, Truck Sales Fell 21.3% in Late April, In Lowest Annual pace Since December
  WSJ: U.S. Car Sales Edged Higher At End of July ––– Auto Makers Keep Making Slow R
  WSJ: Economy: Car Sales Rose Slightly in Latest 10 Days; Greenspan Says Rate Cuts to A
Cluster 4  Size: 16    export international unit japan trade manufacturer citation ger
```

Fig. 3. Keyword representation of clusters is often insufficient to comprehend the meaning of documents in a cluster. Examples come from Lin and Pantel [2002] (left) and Hearst and Pedersen [1996] (right).

a *trimmed sum profile*—an average vector calculated from m documents closest to the cluster's centroid. It should be clear that recovering from such a cluster representation to a semantically comprehensible cluster label is going to be a problem. Scatter/Gather's resolution to it is to construct *cluster digests*—selecting a set of most frequently occurring keywords in the cluster's (or trimmed sum's) documents.

Data-centric algorithms borrow their strengths from well-known and proven techniques targeted at clustering numeric data, but at some point they inevitably hit the problem of *labeling* the output with something sensible to a human. This problem is actually so difficult that the description of a cluster is typically recovered from its feature vector (or centroid) in the simplest possible way, and consists of a few unordered salient terms. The output of the *keyword set* technique is however hardly usable. As an illustration, Figure 3 presents screenshots of results from two data-centric systems, including Scatter/Gather. The whole effort of coming up with an answer as to why the documents have been placed together in a group is thus shifted to the user.

Other examples of data-centric algorithms can be found in systems such as Lassi, WebCat or AIsearch, which will be characterized in Section 5. Another interesting algorithm that is inherently data-centric, but tries to create features comprehensible for users and brings us closer to the next section, is called tolerance rough set clustering (TRSC) [Ngo and Nguyen 2004]. In TRSC, the bag-of-words document representation is enriched with features possibly missing in a document and calculated using a tolerance rough set model computed over the space of all of the collection's terms. After clustering is performed (using traditional K-means or AHC techniques), cluster labels are created by selecting frequent word n-grams from the set of the cluster's documents.

It should be noted that data-centric text clustering algorithms are not necessarily worse than the algorithms in the other two groups (description-aware and description-centric). Their only drawback is the basic problem with cluster labeling: keyword-based representation seems to be insufficient from the user perspective and it makes this class of algorithms less fit for solving our original problem.

4.3.2. Description-Aware Algorithms. The main difficulty in data-centric algorithms is creating a comprehensible description from a model of text representation that is not prepared for this purpose. Description-aware algorithms are *aware* of this labeling problem and try to ensure that the construction of cluster descriptions is that feasible and it yields results interpretable to a human.

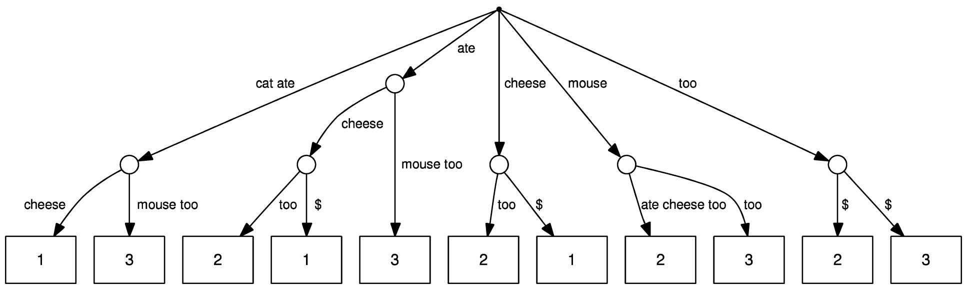


Fig. 4. A generalized suffix tree for three sentences: (1) *cat ate cheese*, (2) *mouse ate cheese too*, and (3) *cat ate mouse too*. Internal nodes (circles) indicate phrases (on the path from the root node) that occurred more than once in original sentences. Leaf nodes (rectangles) show the sentence number. A dollar symbol is used as a unique end-of-sentence marker. Example after Zamir and Etzioni [1998].

One way to achieve this goal is to use a monothetic clustering algorithm (i.e., one in which objects are assigned to clusters based on a single feature) and carefully select the features so that they are immediately recognizable to the user as something meaningful. If features are meaningful and precise then they can be used to describe the output clusters accurately and sufficiently. The algorithm that first implemented this idea was Suffix Tree Clustering (STC), described in a few seminal papers by Zamir and Etzioni [1998, 1999], and implemented in a system called Grouper. In practice, STC was as much of a break through to search results clustering as Scatter/Gather was to the overall concept of using clusters as a text browsing interface. We now introduce its fundamental concepts.

Zamir and Etzioni clearly defined STC's objectives such as an online processing requirement, and focused attention on cluster label descriptiveness. Driven by these requirements they suggested using ordered sequences of words (phrases) as atomic features in addition to isolated words. Not every sequence of words is a good phrase though—phrases should be possibly self-contained (avoid references to other parts of text), complete (should form correct grammatical structures) and meaningful (should provide some information). Searching for candidates for good phrases one could split the text into *chunks* [Abney 1991] and extract chunks with a noun head; these are known to carry most of the information about objects in English. Instead of relying on NLP, however, Zamir and Etzioni used *frequent phrases*—such sequences of words that appear frequently in the clustered documents (and fulfill some additional heuristic rules, for example do not cross a sentence boundary). The ingenious element was in showing how these phrases could be extracted in time linear with the size of the input, using a data structure that had already been known for a long time—a *generalized suffix tree* (GST).

For a sequence of elements $e_0, e_1, e_2, \ldots e_i$; a *suffix* is defined as a subsequence $e_j, e_{j+1}, e_{j+2}, \ldots e_i$, where $j \leq i$. A *suffix tree* for a given sequence of elements contains any of its suffixes on the path from the root to a leaf node. A generalized suffix tree is similar to a suffix tree, but contains suffixes of more than one input sequence with internal nodes storing pointers to original sequences. Note that each internal node of a GST indicates a sequence of elements that occurred at least twice at any position in the original input. Figure 4 illustrates a GST created for three example sentences, with words as basic units in the tree.

The rest of the STC algorithm is fairly simple. STC works in two phases. In the first phase, it tokenizes the input into sentences and inserts words from these sentences into a GST. After all, the input sentences have been added to the suffix tree, the algorithm traverses the tree's internal nodes looking for phrases that occurred a certain number of times in more than one document. Any node exceeding the minimal count of documents and phrase frequency immediately becomes a *base cluster*. Note that so far this clustering is monothetic—the reason for forming a cluster is frequent phrase containment, and this phrase can be immediately used as the base cluster's label. Unfortunately, after the first step STC falls back on a simple merging procedure where base clusters that overlap too much are iteratively combined into larger clusters (a single-link AHC algorithm in the essence). This procedure is not fully justified—it causes the algorithm to become polythetic and may result in chain-merging of base clusters that should not be merged.

We should emphasize that STC made a great impact not because it had a much better quality of document clustering (even if this was originally the authors' intention), but rather because it clearly motivated the need for sensible cluster labels and indicated a way of efficiently constructing such labels directly from features, thus avoiding the difficulties so typical of its data-centric cousins.

As an initial work on the subject, STC left much room for improvement. First, the way it produced a hierarchy of concepts was based on an endlessly transitive relation of document overlap. This issue was addressed in a follow-up algorithm called HSTC [Maslowska 2003]. In HSTC, the simple document overlap criterion in the merge phase is replaced with a dominance relationship between base clusters. The most interesting (topmost) clusters are extracted as a subset of nondominated base clusters (so-called graph kernels).

The second problem with STC was the use of continuous phrases as the only features measuring similarity between documents. This was shown to cause certain problems in languages where the positional order of parts of speech in a sentence may change [Stefanowski and Weiss 2003b]. A follow-up algorithm that tried to overcome this issue was SnakeT, implemented as part of a system with an identical name [Ferragina and Gulli 2004]. SnakeT introduced novel features called approximate sentences—in essence noncontinuous phrases (phrases with possible gaps). SnakeT's authors took their algorithm one step further by expanding the potential set of cluster labels with phrases acquired from a predefined index. Still, clustering precedes and dominates the labeling procedure—something the last group of algorithms tries to balance and sometimes even reverse.

4.3.3. Description-Centric Algorithms. Members of this group include algorithms that are designed specifically for clustering search results and take into account both quality of clustering and descriptions. In fact, quality of the latter is often placed before mere document allocation; if a cluster cannot be described, it is presumably of no value to the user and should be removed from the view entirely. This unorthodox way of placing cluster descriptions before document allocation quality, descends directly from the very specific application type. We believe description-centric algorithms reflect Vivísimo's *description comes first* motto in the closest way.

The description-centric philosophy is omnipresent in the commercial search results clustering systems. Vivísimo has focused on providing sensible cluster descriptions from the very beginning, but others—such as Accumo, Clusterizer or Carrot Search—also quickly recognized the benefits for users resulting from comprehensible, meaningful labels. For example, Accumo states on their Web site that "the quality of cluster titles is crucial to the usability of any clustering system." Interestingly, not many existing research systems can be said to implement this thought in practice. We would like to demonstrate a description-centric algorithm based on the example of Lingo [Osiński et al. 2004; Osiński and Weiss 2005], a search results clustering algorithm implemented in the open source Carrot^2 framework.

Lingo processes the input in four phases: snippets preprocessing, frequent phrase extraction, cluster label induction, and content allocation. The initial two phases are similar to what we described in Section 4.2, with the exception that instead of suffix trees, frequent phrases are extracted using suffix arrays [Manber and Myers 1993]. What makes Lingo different is what happens to frequent phrases after they have been extracted. Recall that frequent phrases in STC served as initial seeds of final clusters, regardless of how much sense they actually made or how commonly they co-occurred with each other. In contrast with that approach, Lingo attempts to identify certain dominating *topics*, called abstract concepts, present in the search results and picks only such frequent phrases that best match these topics.

To discover abstract concepts, the algorithm expresses all documents in the vector space model and puts together term vectors for all documents to form a term-document matrix; let us call it A. The value of a single component of this matrix depends on the strength of the relationship between a document and the given term. Then, singular value decomposition (SVD) is applied to A, breaking it into three matrices: U, S, and V, in such a way that $A = USV^T$, where U contains the left singular vectors of A as columns, V contains right singular vectors of A as columns, and S has singular values of A ordered decreasingly on its diagonal. An interesting property of the SVD decomposition is that the first r columns of matrix U, r being the rank of A, form an orthogonal basis for the term space of the input matrix A. It is commonly believed that base vectors of the decomposed term-document matrix represent an approximation of *topics*—collections of related terms connected by latent relationships. Although this fact is difficult to prove, singular decomposition is widely used in text processing, for example in latent semantic indexing [Deerwester et al. 1990]. From Lingo's point of view, basis vectors (column vectors of matrix U) contain exactly what it has set out to find—a representation of the abstract concepts in the vector space model.

Abstract concepts are quite easy to find, but they are raw sets of term weights still expressed within the vector space model, which as we mentioned, is not very helpful for creating semantic descriptions. Lingo solves this problem in a step called *phrase matching*, which relies on the observation that both abstract concepts and frequent phrases are expressed in the same term vector space—the column space of the original term-document matrix A. Hence, one can calculate the *similarity* between frequent phrases and abstract concepts, choosing the most similar phrases for each abstract concept and thus *approximating* their vector representation with the closest (in the sense of distance in the term vector space) semantic description available. Note that if no frequent phrase can be found for a certain vector, it can be simply discarded—this follows the intuition presented at the beginning of this section. The final step of Lingo is to allocate documents to the frequent phrases selected in the previous step. For each frequent phrase, the algorithm assigns documents that contain it. Compiling everything together, we end up with a monothetic clustering algorithm (document-cluster relationship is determined by label containment), where topics of document groups should be diverse (SVD decomposition), and cluster descriptions comprehensible (because they are phrases extracted directly from the input).

Lingo has shortcomings too—the SVD decomposition is computationally quite demanding, for example. However, the idea of using a data-centric algorithm to determine the model of clusters, followed by a certain process of *label selection* from a set of valid, comprehensible candidates is quite general. Weiss [2006] shows how to modify a traditional data-centric k-Means algorithm to adjust it to the description-centric pattern similar to the one we described here. Other algorithms, utilizing entirely different concepts, but classifiable as description-centric include SRC [Zeng et al. 2004] and DisCover [Kummamuru et al. 2004] (both reviewed in the following), the system presented in Wang and Zhai [2007], which categorizes search results around the topic themes learned from Web search logs, and even to some extent, techniques based on concept lattices, as shown in the CREDO system [Carpineto and Romano 2004a] or in Hotho et al. [2003]. In our opinion, being able to call an algorithm description-centric is not a matter of a specific technique, but rather fulfillment of the objectives concerning cluster labels—comprehensibility, conciseness and transparency of its relationship to the cluster's documents.

4.4. Visualization of Clustered Results

As the clustered results are primarily intended for browsing retrieval, the scheme chosen for visualizing and manipulating the hierarchy is very important. The largely predominant approach is based on hierarchical folders, illustrated previously in Figure 1. The system displays the labels of the top level of the cluster hierarchy using an indented textual representation, and the user may click on each label to see the snippets associated with it as well as to expand its subclusters, if any. The user can then repeat the same operation on the newly displayed (sub-)clusters.

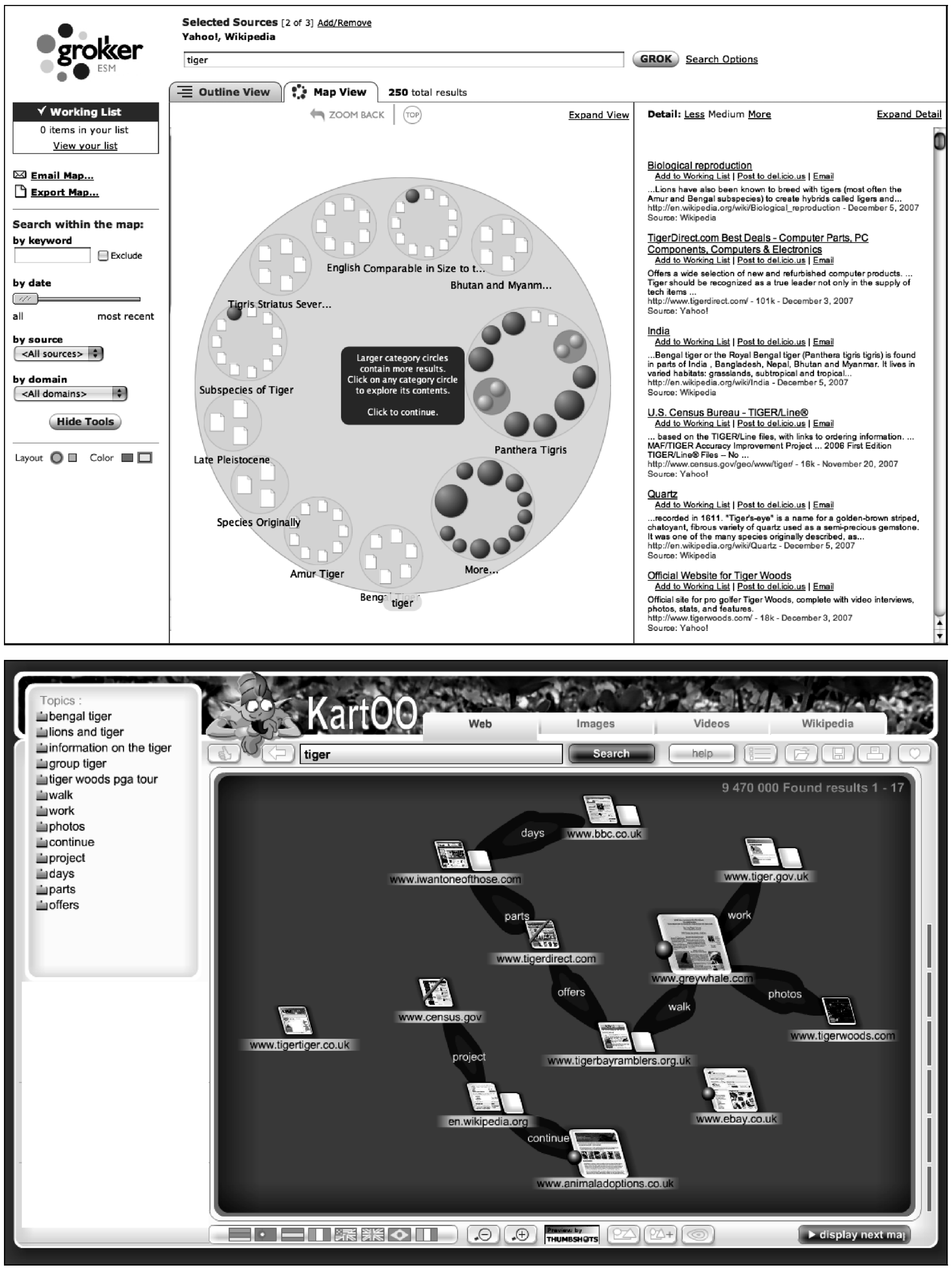
Usually, the most populated clusters are shown earlier and the clusters with very few snippets are displayed on demand. Furthermore, all the snippets of one cluster that are not covered by its displayed children are grouped into a dummy concept named "other." Figure 1 is an example in which only the top clusters of the top level of the hierarchy are shown, and in which the user chose to see the documents associated with the cluster "mac" without expanding its subclusters.

The folder-tree display has been successfully adopted by Vivísimo and by many other systems, including Carrot[2], CREDO, and SnakeT. As the metaphor of hierarchical folders is used for storing and retrieving files, bookmarks, menus items, and so on, most users are familiar with it and hence no training is required. Furthermore, the textual folder hierarchy layout is very efficient to compute and easy to manipulate.

On the other hand, a textual indented representation is probably neither the most compact nor the most visually pleasing tree layout. Through a graphical display, the relationships in size, distance, and kind (folder or search results) between the clusters can be rendered by rich spatial properties such as dimension, color, shape, orientation, enclosure, and adjacency. The research on information visualization has explored a huge number of techniques for representing trees. Two excellent reviews of this work are Herman et al. [2000] and Katifori et al. [2007], the latter with a focus on information retrieval.

In practice however, only a few alternative tree visualization schemes to the folder layout have been used in the deployed clustering engines. The most notable system is Grokker[3]. Taking a *nesting and zooming* approach, Grokker displays each cluster as a circle and its subclusters as smaller circles within it (see Figure 5). The system allows the user to zoom in on the child nodes, making them the current viewing level. Another clustering engine that made use of an elaborate tree visualization technique is Lassi [Maarek et al. 2000]. It provided the user with a fish-eye view, which makes it possible to combine a focus node with the whole surrounding context through a distorted representation [Sarkar and Brown 1994]. Simpler nonstandard approaches have been taken in Grouper [Zamir and Etzioni 1999], where just one level of the hierarchy was represented, using a raw table, and Grokker Text, which displayed the results in a manner similar to Mac OS X's Finder.

[3]www.groker.com



Biological reproduction
Add to Working List | Post to del.icio.us | Email
...Lions have also been known to breed with tigers (most often the Amur and Bengal subspecies) to create hybrids called ligers and...
http://en.wikipedia.org/wiki/Biological_reproduction - December 5, 2007
Source: Wikipedia

TigerDirect.com Best Deals - Computer Parts, PC Components, Computers & Electronics
Add to Working List | Post to del.icio.us | Email
Offers a wide selection of new and refurbished computer products. ... Tiger should be recognized as a true leader not only in the supply of tech items ...
http://www.tigerdirect.com/ - 101k - December 3, 2007
Source: Yahoo!

India
Add to Working List | Post to del.icio.us | Email
...Bengal tiger or the Royal Bengal tiger (Panthera tigris tigris) is found in parts of India , Bangladesh, Nepal, Bhutan and Myanmar. It lives in varied habitats: grasslands, subtropical and tropical...
http://en.wikipedia.org/wiki/India - December 5, 2007
Source: Wikipedia

U.S. Census Bureau - TIGER/Line®
Add to Working List | Post to del.icio.us | Email
... based on the TIGER/Line files, with links to ordering information. ... MAF/TIGER Accuracy Improvement Project ... 2006 First Edition TIGER/Line® Files – No ...
http://www.census.gov/geo/www/tiger/ - 16k - November 20, 2007
Source: Yahoo!

Quartz
Add to Working List | Post to del.icio.us | Email
...recorded in 1611. "Tiger's-eye" is a name for a golden-brown striped, chatoyant, fibrous variety of quartz used as a semi-precious gemstone. It was one of the many species originally described, as...
http://en.wikipedia.org/wiki/Quartz - December 5, 2007
Source: Wikipedia

Official Website for Tiger Woods
Add to Working List | Post to del.icio.us | Email
Official site for pro golfer Tiger Woods, complete with video interviews, photos, stats, and features.
http://www.tigerwoods.com/ - 18k - December 3, 2007
Source: Yahoo!

Fig. 5. Clustered search results for the query "tiger." Screenshots from the commercial engines Grokker and KartOO.

The clustering engines that do not follow a strict hierarchical organization can still use a tree-based visualization, or they can adopt different strategies. CREDO, for instance, builds a lattice structure that is subsequently transformed into a redundant cluster tree for visualization purposes. By contrast, other systems use a truly graph-based interface. The best known example is probably KartOO[4], shown in Figure 5. The results contained in a cluster are represented as a graph, in which the nodes are individual results represented by their thumbnail and URL, the (sub)clusters are represented as virtual regions of the graph, and the edges are keywords shared by the nodes. This information is usually not available in the other clustering engines, but it comes at the cost of hiding the detailed content given in titles and snippets.

The system named WhatsOnWeb [Di Giacomo et al. 2007] further extends the graph-based visualization paradigm by also displaying relationships between clusters. By combining a graph drawing algorithm [Eades and Tamassia 1989] with a *nesting and zooming* interaction paradigm, WhatsOnWeb displays an expandable map of the graph of clusters, which preserves the shape of the drawing when the user clicks on some node in the map. The system also supports different layouts: radial [Stasko and Zhang 2000] and Treemaps [Johnson and Shneiderman 1991]. Similar to KartOO, WhatsOnWeb does not show the textual content of search results.

Among the factors that may have hampered the deployment of more sophisticated visualization schemes, one can consider their lack of friendliness, due to the inherent complexity of some visualization metaphors, as well as the limited practical availability of advanced displays, interactive facilities, and bandwidth. The ease of understanding and manipulation on the part of the user indeed seems the most critical issue. A recent comparison of textual and zoomable interfaces to search results clustering has shown that even if the raw retrieval performance of the tested systems was comparable, users subjectively preferred Vivísimo's textual interface [Rivadeneira and Bederson 2003]. In another experiment [Turetken and Sharda 2005], a TreeMap visualization of clustered results was found to be more effective than a ranked list presentation. Unfortunately, in this latter experiment the effect of clustering was not decoupled from that of the visual interface.

A powerful visualization technique that goes beyond the capabilities of the existing interfaces to Web clustering engines is the combination of multiple partial views, although in our case it requires the ability to partition the document descriptors in useful subsets (or facets). For example, when the data can be classified along orthogonal dimensions (e.g., functional, geographical, descriptive), it may be convenient for the user to choose the descriptors in an incremental fashion, making decisions based on the information displayed by the system for the current choice of the descriptors. This technique has been explored in different fields and implemented in various guises (e.g., Roberts [1998]; Hearst et al. [2002]; Cole et al. [2003]; Hearst [2006]), and it may be worth investigating even for visualizing search results clustering, especially for some types of data and search tasks. An interesting discussion of emerging interface designs that allow flexible navigation orders for the cases when a hierarchy's content is inherently faceted is presented in Perugini and Ramakrishnan [2006].

5. OVERVIEW OF WEB CLUSTERING ENGINES

In the following, we briefly review the most influential search results clustering systems that have appeared in literature or have made a commercial impact. We make a distinction between research and commercial systems, even if such a separation may sometimes be difficult (SRC authors affiliated with Microsoft, DisCover authors affiliated

[4]www.kartoo.com

Table III. A Summary of Research Search Results Clustering systems. URLs to Last-Known Demo Location is Given Below the Table

System name (algorithm alias)	Year	Text Features	Cluster Labels	Clustering Method	On-line Demo	Clusters Structure	Source Code
Grouper (STC)	1998	single words, phrases	phrases	STC	yes (dead)	flat, concept cloud	no
Lassi	2000	lexical affinities	lexical affinities	AHC	no (desktop)	hierarchy	no
CIIRarchies	2001	single words	word sets	language model/ graph analysis	yes (dead)	hierarchy	no
WICE (SHOC)	2002	single words, phrases	phrases	SHOC	yes (dead)	hierarchy	no
Carrot^2 (Lingo)	2003	frequent phrases	phrases	Lingo	yes	flat	yes
Carrot^2 (TRSC)	2004	words, tolerance rough sets	n-grams (of words)	TRSC	yes	flat (optional hierarchy)	yes
WebCat	2003	single words	words	k-Means	yes (dead)	flat	no
AISearch	2004	single words	word sets	AHC + weighted centroid covering	yes (dead)	hierarchy	no
CREDO	2004	single words	word sets	concept lattice	yes	graph	no
DisCover	2004	single words, noun phrases	phrases	incremental cove-rage optimization	no	hierarchy	no
SnakeT	2004	approximate sentences	phrases	approx. sent. coverage	yes	hierarchy	no
SRC	2004	n-grams (of words)	n-grams (of words)	SRC	yes	flat (paper) hierarchy (demo)	no
EigenCluster	2005	single words	three salient terms	divide-merge (hybrid)	yes	flat (optional hierarchy)	no
WhatsOnWeb	2006	single words	phrases	edge connectivity	yes	graph	no

CIIRarchies: http://www.cs.loyola.edu/∼lawrie/hierarchies/, WebCAT: http://ercolino.isti.cnr.it/webcat/, Carrot^2: http://www.carrot2.org, AISearch: http://www.aisearch.de, Credo: http://credo.fub.it, SnakeT: http://snaket.di.unipi.it, SRC: http://rwsm.directtaps.net, EigenCluster: http://eigencluster.csail.mit.edu, WhatsOnWeb: http://gdv.diei.unipg.it/view/tool.php?id=wow.

with IBM). Descriptions of research systems are ordered according to the approximate date of first appearance (article publication, where applicable), commercial systems are ordered alphabetically. Neither list is exhaustive.

5.1. Research Systems

The following list of research systems contains backreferences whenever the system was previously discussed in this article. A summarized table of algorithmic features and Web links to online demonstrations of systems (whether still functional or not) is given in Table III. The computational complexity of the algorithms (exact or estimated) is reported if available.

Grouper (STC). We described Grouper [Zamir and Etzioni 1999] in Section 4.3.2 while talking about the STC algorithm.

Lassi. In Lassi (an acronym for 'librarian assistant'), the authors used a traditional agglomerative hierarchical clustering algorithm, but they improved the feature selection phase [Maarek et al. 2000]. Instead of single words, pairs of words with strong correlation of appearance in the input text are used (such pairs are said to share a *lexical affinity*). An index of lexical affinities discovered in the input is later reused for labeling the discovered clusters. The computational complexity of the algorithm is $O(n^2)$, where n is the number of documents to be clustered.

CIIRarchies. CIIRarchies produces a hierarchy of concepts found in a document collection by employing a probabilistic language model and graph analysis [Lawrie et al. 2001; Lawrie and Croft 2003]. The algorithm first builds a graph of terms connected by edges with weights indicating conditional probabilities between a given pair of words. Then a heuristic is used to detect dominating sets of vertices (*topic terms*) with high vocabulary coverage and maximum joint conditional probabilities. The algorithm then descends recursively to produce a hierarchy.

Even though CIIRarchies does not utilize a clustering algorithm directly, it is functionally equivalent to a description-centric algorithm because clusters are formed through the perspective of their future descriptions (unfortunately these consist of single terms).

WICE (SHOC). Suffix trees used in the STC algorithm are known to be problematic in practical implementations due to non-locality of data pointers virtually jumping across the data structure being built. The authors of WICE (Web information clustering engine) devise an algorithm called SHOC (semantic hierarchical online clustering) that handles data locality and successfully deals with large alphabets (languages such as Chinese). For solving the first problem, SHOC [Dong 2002; Zhang and Dong 2004] uses suffix arrays instead of suffix trees for extracting frequent phrases. Then, the algorithm builds a term-document matrix with columns and rows representing the input documents and frequent terms/phrases, respectively. Using singular value decomposition (SVD), the term-document matrix is decomposed into two orthogonal matrices, U and V, and the matrix of singular values S (see Section 4.3.3 for more information about SVD decomposition). Based on the S matrix, SHOC calculates the number of clusters to be created. The V matrix determines which documents belong to each cluster and the largest element of each column of matrix U determines which frequent term or phrase should be used to label the corresponding cluster. The fact that the U and V matrices are orthogonal helps to ensure that clusters represent groups of dissimilar documents.

Note that SHOC is similar to the Lingo algorithm, although it lacks the safety vent resulting from separate cluster label assignment and document allocation phases—all base vectors of matrix U become clusters, regardless of whether a close enough frequent phrase could be found to describe it.

WebCAT. WebCAT [Giannotti et al. 2003], developed at the Pisa KDD Laboratory, was built around an algorithm for clustering categorical data called *transactional k-Means*. Originally developed for databases, this algorithm has little to do with transactional processing and is rather about careful definition of (dis)similarity (Jaccard coefficient) between objects (documents) described by categorical features (words). WebCAT's version of transactional k-Means assumes that the more features (words) two documents have in common, the more they are similar. A cluster is formed by finding documents with at least τ shared words, where τ is called *confidence*. WebCAT's computational complexity is linear in the number of documents to be clustered, assuming a fixed number of iterations. An algorithm with a very similar principle, but different implementation, was presented concurrently in Pantel and Lin [2002] and was adequately called *clustering with committees*.

Carrot^2. Carrot^2 combines several search results clustering algorithms: STC, Lingo, TRSC, clustering based on swarm intelligence (ant-colonies), and simple agglomerative techniques. We described Carrot^2 and Lingo in Section 4.3.3, so we omit its further description here.

Of some interest is the fact that committers affiliated with Carrot^2 also started a spin-off company called Carrot Search[5] and offer a commercial successor—a clustering engine called Lingo3G. Despite similar names, Lingo and Lingo3G are two completely different clustering algorithms. While Lingo uses SVD as the primary mechanism for cluster label induction, Lingo3G employs a custom-built metaheuristic algorithm that aims to select well-formed and diverse cluster labels. In contrary to Lingo, Lingo3G creates hierarchical clusters.

AISearch. A different cluster labeling procedure is shown in the weighted centroid covering algorithm [Stein and Meyer zu Eissen 2004], used in the AISearch system. The authors start from a representation of clusters (centroids of document groups) and then build their word-based descriptions by iterative assignment of highest scoring terms to each category, making sure each unique term is assigned only once. The computational complexity of the algorithm is $O(m \log_2 m)$, where m is the number of terms. Interestingly, the authors point out that this kind of procedure could be extended to use existing ontologies and labels, but the papers provide no insight as to whether this idea has been integrated in the public demo of the system.

CREDO. In CREDO [Carpineto and Romano 2004a, 2004b], the authors utilize formal concept analysis [Ganter and Wille 1999] to build a lattice of concepts (clusters) later presented to the user as a navigable tree. The algorithm works in two phases. In the first phase, only the titles of input documents are taken into account to generate the most general concepts (to minimize the risk of proliferation of accidental combinations of terms at the top level of the hierarchy). In the second phase, the concept lattice-based method is recursively applied to lower levels using a broader input of both titles and snippets. The computational complexity of the algorithm is $O(nmC + m)$, where n is the number of documents, m is the number of terms, and C is the number of clusters. Although the system uses single words as features of search results, it can produce multiple-word cluster labels reflecting the presence of deterministic or causal associations between words in the query context.

CREDO is an interesting example of a system that attempts to build the taxonomy of topics and their descriptions simultaneously. There is no explicit separation between clustering (concept formation) and labeling—these two elements are intertwined to produce comprehensible and useful output. Another feature of CREDO is that the structure is a lattice instead of a tree. The navigation is thus more flexible because there are multiple paths to the same node, but the presence of repeated labels along different paths may clutter the layout.

DisCover. In Kummamuru et al. [2004], the authors present a system called DisCover. The algorithm inside it works by defining an optimality criterion and progressively selecting new concepts (features) from a set of candidates at each level of the hierarchy. The selection process attempts to maximize coverage (defined as the number of documents covered by a concept) and maintain distinctiveness between the candidate concept and other concepts on the same branch of the hierarchy. Concepts are selected from a set of noun phrases and salient keywords, so the resulting algorithm

[5]www.carrot-search.com

could be classified as description-centric. However, from the presented screenshots it appears that cluster labels are usually single terms and hence their descriptive power can be questioned. The computational complexity of DisCover is $O(mC)$, where m is the number of features and C is the number of clusters.

SnakeT. SnakeT [Ferragina and Gulli 2004, 2005] is both the name of the system and the underlying algorithm. We described its main characteristics in Section 4.3.2. An additional interesting feature of SnakeT is that it builds a hierarchy of possibly overlapping folders. The computational complexity of the algorithm is $O(n \log_2 n + m \log_2 mp)$, where n is the number of documents, m is the number of features, and p is the number of labels.

SRC. Developed at Microsoft Asia, SRC is an add-on to MSN Search. The algorithm behind SRC [Zeng et al. 2004] is an interesting attempt to create a cluster label selection procedure that is trained on real data. This way the authors replace an unsupervised clustering problem with a supervised classification problem (selection of the best label candidates according to a known preference profile). In the first step, a set of fixed-length word sequences (3-grams) is created from the input. Each label is then scored with an aggregative formula combining several factors: phrase frequency, length, intra-cluster similarity, entropy, and phrase independence. The specific weights and scores for each of these factors were learned from examples of manually prioritized cluster labels. Experimentally, the computationally complexity of the algorithm grew linearly with the number of documents.

EigenCluster. EigenCluster is a clustering system optimized for efficiency and scalability. The algorithm employed by EigenCluster, called divide-and-merge [Cheng et al. 2005], is a combination of spectral clustering (efficient with sparse term-document matrices) and dynamic programming (for merging nodes of the tree resulting from the divide phase). The spectral clustering part is optimized so as to preserve sparsity of the input data. If each cut through the term-document matrix is balanced, the complexity of the algorithm is bounded by $O(M \log_2 n)$, where M is the number of non-zero elements in the term-document matrix, and n is the number of documents. The authors show that this theoretical bound is even lower on real-life data sets.

Even though EigenCluster is a relatively new system, it sticks to very simple cluster labels. In the online demonstration, each cluster is labeled with three salient keywords; phrases or more complex structures are not used.

WhatsOnWeb. WhatsOnWeb [Di Giacomo et al. 2007] explicitly addresses the problem of finding relationships between subtopics that are not captured by the ancestor/descendant relation. The graph of clusters is computed in two steps. First, a *snippet graph* is constructed using the number and the relevance of sentences (consecutive stems) shared between the search results. Second, the clusters and their relationships are derived from the snippet graph by finding the vertices that are most strongly connected. Such a clustered graph is then pruned and displayed using an elaborate graphical interface, as reviewed in 4.4.

WhatsOnWeb is also interesting because it is one of the few Web clustering engines rooted in graph theory, together with Wang and Zhai [2007]. One drawback of the system is the high response times required for processing a query (of the order of tens of seconds), mainly due to the inefficiency of computing the clusters from the snippet graph.

Table IV. Commercial Companies Offering Technologies for Clustering Search Results

Name	Company	Cluster labels	URL	Clusters structure
Accumo	Accumo	Phrases	www.accumo.com	Tree
Clusterizer	CyberTavern	Phrases	www.iboogie.com	Tree
Cowskid	Compara	Terms	www.cowskid.com	Flat
Fluster	Funnelback	Phrases	www.funnelback.com	Flat
Grokker	Grokker	Phrases	www.grokker.com	Graphical/Tree
KartOO	KartOO	Phrases	www.kartoo.com	Graphical/Tree
Lingo3G	Carrot Search	Phrases	www.carrot-search.com	Tree
Mooter	Mooter Media	Phrases	www.mooter.com	Graphical/Flat
WebClust	WebClust	Phrases	www.webclust.com	Tree
Vivísimo	Vivísimo	Phrases	www.vivisimo.com	Tree

5.2. Commercial Systems

A number of commercial systems offering postretrieval clustering exist but not much is known with regard to the techniques they employ. The most broadly recognized and cited commercial company specializing in clustering technology is Vivísimo, standing behind a clustering search engine, Clusty (we list a few other commercial clustering engines in Table IV).

Recently a number of companies (Google, Gigablast, Yahoo!) have been experimenting with various query-refinement and suggestion techniques. We decided not to list them in Table IV because it is not known whether this functionality comes from a postretrieval clustering algorithm or is achieved in a different way (e.g., by mining query logs).

6. EFFICIENCY AND IMPLEMENTATION ISSUES

As the primary aim of a search results clustering engine is to decrease the effort required to find relevant information, user experience of clustering-enabled search engines is of crucial importance. Part of this experience is the speed at which the results are delivered to the user. Ideally, clustering should not introduce a noticeable delay to normal query processing. In the following subsections we describe the factors that contribute to the overall efficiency and summarize a number of techniques that can be used to increase user-perceived performance.

6.1. Search Results Clustering Efficiency Factors

Several factors contribute to the overall computational performance of a search results clustering engine. The most critical tasks involve the first three components presented in Figure 2, namely search result acquisition, preprocessing, and clustering. The visualization component is not likely to affect the overall system efficiency in a significant manner.

6.1.1. Search Results Acquisition. No matter whether search results are fetched from a public search engine (e.g., through Yahoo! API) or from a dedicated document index, collecting input for clustering amounts to a large part of the total processing time.

In the first case, the primary reason for the delay is the fact that the number of search results required for clustering (e.g., 200) cannot be fetched in one remote request. The Yahoo! API allows up to 50 search results to be retrieved in one request, while Google SOAP API returns a mere 10 results per one remote call. Issuing a number of search requests in parallel can decrease the total waiting time, but, as shown in Table V, the delays still fall in the 1–6 second range.

Table V.
Time required to fetch 200 search results from various sources. The experiment was performed on a computer with a broadband network access; the results are averaged over 40 samples. Different queries were used for each sample to avoid cache influence. Fetching was performed in parallel requests if there was a cap for maximum results per query, Lucene Index was stored on a local hard drive

Source	Avg. delay [s]	Std. dev. [s]
Yahoo! API	2.12	0.65
Google API	5.85	2.35
MSN API	0.56	0.43
Lucene (snippets)	1.78	0.50

When fetching search results from a dedicated document index, the overhead related to network communication can be avoided. In this scenario, however, the document retrieval engine will need to extract the required number of documents from its internal storage, which can be equally time-consuming. Additionally, the engine may also need to create contextual document snippets, which will increase the processing time even further. Table V shows the time required to fetch 200 search results from a popular open source document retrieval engine, Lucene, compared to the same number of search results fetched from commercial search engines using their respective APIs. The times reported in Table V include snippet creation; the index was stored on a local hard drive. The results obviously depend on network congestion (remote APIs), on the capability of local equipment used (Lucene), and also on the specific server processing the request on the search engine side, but are generally representative of commodity hardware and typical network locations.

6.1.2. Clustering. Depending on the specific algorithm used, the clustering phase can significantly contribute to the overall processing time. Although most of the clustering algorithms will have common components, such as input tokenization and stemming, the efficiency of clustering is mainly a function of the computational complexity of the core clustering element of a particular algorithm. The clustering component may in turn comprise several steps.

In the case of the Lingo algorithm, for example, the most time consuming operation is computing the SVD decomposition of the term-document matrix with a computational complexity equal to $O(m^2n + n^3)$, for a $m \times n$ matrix, where in the case of Lingo, n is the number of search results being clustered, and m is the number of features used for clustering. While this may seem costly, search results clustering is a very specific domain, where problem instances rarely exceed a few hundred snippets (it is impractical to fetch more than, say 500 search results, because of the network latencies). Search results clustering systems must therefore be optimized to handle smaller instances and process them as fast as possible. This assumption to some degree justifies the use of techniques that would be unacceptable in other fields due to scalability issues.

In production-quality clustering the system efficiency of the physical implementation turns out to be just as important as the theoretical complexity of the algorithm. A canonical example is frequent phrase extraction. In theory, Esko Ukkonen's algorithm [Ukkonen 1995] builds a suffix tree for a sequence of n elements with the cost of $O(n)$. In practice, the algorithm makes very inefficient use of memory caches and, in the worst case when swap memory must be used, its performance degrades to an unacceptable level. Suffix arrays, mentioned previously, have a higher theoretical complexity of $O(n \log n)$, but are much more efficient in practice due to the locality of data structure updates.

Table VI.
Time required to cluster 50, 100, 200 and 400 search results using several algorithms discussed in this survey: CREDO, Lingo, Lingo3G, Suffix Tree Clustering (STC), and Tolerance Rough Set Clustering (TRSC). All algorithms used their default parameter and threshold settings. The benchmark clustered search results retrieved from the Yahoo! API for query "London". For each dataset size and algorithm, we provide the time averaged across 75 runs (preceded by 25 untimed warm-up runs to allow Internal Java virtual machine optimizations to take place).

Algorithm	50 snippets [s]	100 snippets [s]	200 snippets [s]	400 snippets [s]
CREDO	0.031	0.088	0.272	0.906
Lingo	0.025	0.138	0.243	0.243
Lingo3G	0.009	0.020	0.045	0.070
STC	0.007	0.014	0.030	0.070
TRSC	0.072	0.552	1.368	4.754

Another example concerns the performance of tokenization. Tokenizers will have a different performance characteristic depending on whether they were hand-written or automatically generated. Furthermore, the speed of automatically generated tokenizers (for the same grammar!) may vary greatly. According to the benchmarks authors of this paper performed while implementing the Carrot^2 framework,[6] tokenizers generated by the JavaCC parser generator were up to 10 times slower than equivalent tokenizers generated using the JFlex package.

To give an impression of the processing effort involved in search results clustering, Table VI provides clustering times for several algorithms discussed in this article. We chose these systems because they were available to us for testing and they are representative of the main categories discussed in this article. The times, with the exception of TRSC, are all below one second, which makes such algorithms suitable for online processing. Lingo imposes a limit on the maximum size of the term-document matrix on which it then performs singular value decomposition. Therefore, the times required to cluster 200 and 400 snippets are similar, at the cost of lower clustering quality in case of the larger data set.

Concluding the performance discussion, let us again emphasize that the efficiency of search results clustering is the sum of the time required to fetch the input from a search engine and the time to cluster this input. For the Lingo algorithm, the time spent on cluster construction accounts for about 10–30% of the total processing time (depending on whether a local index or remote data source was used).

6.2. Improving Efficiency of Search Results Clustering

There are a number of techniques that can be used to further improve the actual and user-perceived computational performance of a search results clustering engine.

6.2.1. Client-Side Processing. The majority of currently available search clustering engines assume server-side processing—both search results acquisition and clustering take place on one or more servers. One possible problem with this approach is that during high query rate periods the response times can significantly increase and thus degrade the user experience.

The commercial clustering engine Grokker took a different strategy whereby the search results clustering engine was delivered to the user as a downloadable desktop

[6]All benchmarks related to the Carrot^2 framework we present hereafter were performed on a Windows XP machine with a Core2 Duo 2.0 GHz processor, 2 GB RAM and Sun Server JVM 1.6.x with options: `-Xmx512m -Xms128m -XX:NewRatio=1 -server`.

application.[7] In this scenario, both search results acquisition and clustering would use the client machine's resources: network bandwidth and processor power, respectively. In this way, scalability issues and the resulting problems could be avoided, at the cost however, of for example, the need for initial application setup and the risk of the proprietary implementation of a clustering algorithm to be reverse engineered.

6.2.2. Incremental Processing. As postulated by Zamir and Etzioni [1999], one desirable feature of search results clustering would be incremental processing—the ability to deliver some, possibly incomplete, set of clusters to the users before fetching of the search results has finished. Although incremental text clustering is especially useful in those scenarios where documents arrive continuously, such as news stories, Usenet postings, and blogs [Sahoo et al. 2006], it can also be used for improving the efficiency of Web clustering engines (by halting the computation as soon as some time limit is exceeded).

Some systems employ clustering algorithms that are incremental in nature, including STC and the *Self Splitting Competitive Learning* algorithm [Zhang and Liu 2004]. The latter is especially interesting because it dynamically adjusts the number of clusters to the number of search results seen. Another fairly straightforward form of incremental processing can be achieved with iterative clustering algorithms (e.g., k-means), whereby a less accurate but faster solution can be obtained by halting the algorithm before it reaches the termination condition. However, to the best of our knowledge, none of the publicly available search results clustering engines offer full incremental processing as a whole, for example, including the appropriate user interface.

Some sort of substitute of incremental behavior is used in the Carrot^2 clustering engine, which presents search results as they are being collected and outputs the clusters after all search results have been downloaded. This is, however, an engineering trick and not a full answer to the problem.

6.2.3. Pretokenized Documents. If the Web clustering engine can access the low-level data structures of its underlying search engine[8] and the index contains information about the positions of tokens within documents, the clustering algorithm could fetch tokenized documents straight from the index and save some of the time it would normally have to spend on splitting the documents into tokens. According to the authors' experiences with the Carrot^2 framework, tokenization accounts for about 5–7% of the total clustering time, which in some cases may be enough of an improvement to justify the use of pretokenized documents.

The price to be paid for this kind of optimization is a very tight coupling between the source of search results and the clustering algorithm. Furthermore, the characteristics of the tokenizer used by the search engine may not be fully in line with the requirements of the clustering algorithm. For example, the search engine may bring all tokens to lower case, while the clustering algorithm may prefer receiving tokens in their original form for better presentation of cluster labels, especially those containing acronyms and other nonstandard tokens.

7. EVALUATION OF RETRIEVAL PERFORMANCE

In this section we address the issue of evaluating the retrieval performance of a clustering engine. For the sake of simplicity, we split the discussion in two parts. In the first

[7]This particular version of Grokker seems to be no longer available online.
[8]To the best of our knowledge, HTTP-based APIs for public search engines do not expose tokenized versions of search results.

we consider how to compare the retrieval effectiveness of search results clustering to that of the ranked list of results without clustering. In the second we consider comparing alternative clustering engines. The first task is more difficult because it involves a comparison between two heterogeneous representations: a list of clusters of results versus a list of plain results.

7.1. Search Results Clustering Versus Ranked Lists

Since clustering engines are meant to overcome the limitations of plain search engines, we need to evaluate whether the use of clustered results does yield a gain in retrieval performance over ranked lists.

The typical strategy is to rely on the standard approach developed in the information retrieval field for evaluating retrieval effectiveness, which assumes the availability of a test collection of documents with associated relevance judgments and employs performance measures based on the two notions of *recall* (the ability of the system to retrieve all relevant documents) and *precision* (the ability of the system to retrieve only relevant documents). However, these measures assume that the results are presented in ranked order; in our case, their application requires a preliminary transformation of the clustered results into a plain list of results.

One obvious way to perform such a clustering *linearization* would be to preserve the order in which clusters are presented and just expand their content, but this would amount to ignoring the role played by the user in the choice of the clusters to be expanded. In practice, therefore, some more informed usage assumption is made. One of the earliest and simplest linearization techniques is to assume that the user can choose the cluster with the highest density of relevant documents and to consider only the documents contained in it ranked in order of relevance [Hearst and Pedersen 1996; Schütze and Silverstein 1997]. A refinement of this method [Zamir and Etzioni 1998] consists of casting the output of clustering engines as a clustered list, assuming that the user can pick more than one optimal cluster, or even a fraction of it, when the cluster contains many documents. In practice, however, the user may easily fail to choose the optimal cluster(s) according to this definition.

The clustered list metaphor may be used to reorder the documents using a different method. One can assume that the user goes through the list of clusters and switches over from the current cluster to the next one as soon she sees that the number of non-relevant documents contained in the current cluster exceeds the number of relevant documents. Leuski [2001] suggested that this simple interactive method may be rather effective.

A more analytic approach [Kummamuru et al. 2004], is based on the *reach time*: a modelization of the time taken to locate a relevant document in the hierarchy. It is assumed that, starting from the root, a user chooses one node at each level of the hierarchy by looking at the descriptions of that level, and iteratively drills down the hierarchy until a leaf node that contains the relevant document is reached. At this point, the documents under the leaf node are sequentially inspected to retrieve the relevant document.

Let s be the branching factor, d the number of levels that must be inspected, and $p_{i,c}$; the position of the i-th relevant document in the leaf node. The reach time for the i-th document is: $rt_{clustering} = s \cdot d + p_{i,c}$. Clearly, the corresponding reach time for the ranked list is (where $p_{i,r}$ is the position of the i-th relevant document in the ranked list): $rt_{rankedlist} = p_{i,r}$. The reach times are then averaged over the set of relevant documents.

The reach time can be extended to the situation in which we want to assess the ability of the system to retrieve documents that cover many different subtopics of the given user query, rather than documents relevant to the query as a whole. This task seems very suitable for clustering engines because the subtopics should match the high-level clusters. The measure, called *subtopic reach time* [Carpineto et al. 2009], is defined as the mean, averaged over the query's subtopics, of the smallest of the reach times associated with each subtopic's relevant results. It can be applied to both cluster hierarchies and ranked lists.

If one is interested in maximizing recall rather than precision, it may be more useful to consider how many nodes and snippets must be inspected to collect *all* the relevant docs, rather than assuming that each relevant doc is retrieved in isolation [Cigarrán et al. 2005]. This may more accurately model actual user behavior because the search does not need to start again at the top of the hierarchy after each relevant document has been retrieved.

Whether we aim at precision or recall, it seems that taking explicitly into account, both the nodes that must be traversed and the snippets that must be read—as with the reach time measures—is a better way of measuring the browsing retrieval effectiveness. However, even this approach has drawbacks, because the cognitive effort required to read a cluster description may be different from that required to read a document. On the other hand, it is not guaranteed that a cluster label will allow the user to correctly evaluate the contents of the documents contained in the cluster. Note that both the reach time and subtopic reach time assume that the user is always able to choose the optimal cluster, regardless of its label; in this sense, such measures provide an upper bound of the true retrieval performance. The quality of the cluster labels is, in practice, a key to high retrieval performance. This issue will be discussed in the next section.

The need for a test collection with specified relevance judgments can be avoided by analyzing user logs. Zamir and Etzioni [1999] compared search engine logs to clustering engine logs, computing several metrics such as the number of documents followed, the time spent, and the click distance. The interpretation of user logs is, however, difficult, because such data usually refers to different users and search tasks and some modeling of users' actions is still required.

Although in this section, we have focused on automatic evaluation methods, it must be noted that conducting user studies is a viable alternative or complementary approach [Chen and Dumais 2000; Turetken and Sharda 2005; Ferragina and Gulli 2005; Käki 2005; and Carpineto et al. 2006]. The user performs some kind of information-seeking task with the systems being compared, the user session is recorded, and the retrieval performance is typically evaluated measuring the accuracy with which the task has been performed, and its completion time. Such studies are especially useful to evaluate inherently subjective features or to gain insights about the overall utility of the methods being tested. The main disadvantage of interactive evaluation methods is that the results may be ambiguous due to the presence of hidden factors, and cannot be replicated.

The experimental findings reported in the literature are in general favorable to clustering engines, suggesting that they may be more effective than plain search engines both in terms of the amount of relevant information found and the time necessary to find it. However, we must be cautious because these results may be biased by unrealistic usage assumptions or unrepresentative search tasks. To date, the evaluation issue has probably not yet received sufficient attention and there is still a lack of conclusive experimental findings for demonstrating the superior retrieval performance of clustering engines over search engines, at least for some type of queries. This may also have been one of the reasons why the major commercial search engines have so far been reluctant to fully embrace the search result clustering paradigm.

7.2. Comparing Alternative Clustering Engines

As there are so many clustering engines, it is important to devise methods for comparing their performance. The most straightforward approach is to use the methods developed in the clustering literature to evaluate the *validity* of search results hierarchies. One can distinguish between internal and external validity [Jain and Dubes 1988], [Halkidi et al. 2001].

Internal validity measures assess certain structural properties of clustering such as cluster homogeneity, topic separation, and outlier detection. Such measures, which have also been widely used for evaluating search results hierarchies, have the advantage that they are based solely on the features that describe the objects to be grouped. However, the fact that a hierarchy has certain structural properties does not guarantee that it is of interest to the user. In many applications, including document clustering, it may be more appropriate to evaluate cluster validity based on how well the clusters generated by the system agree with the ground truth clusters generated by human experts.

This is usually done by adopting a document categorization viewpoint [Sebastiani 2002], which consists of measuring the classification accuracy of the clustering algorithm relative to a particular set of objects that have previously been manually assigned to specific classes. The Open Directory Project, already mentioned, and the Reuters collection[9] are two common test datasets used for evaluating classification accuracy, with the F-measure probably being the best known performance measure. If the correct classification is a hierarchy rather than a partition, it may be necessary to flatten the results of the hierarchy or to consider the clusters at some specified depth.

The amount of agreement between the partition generated by the clustering algorithm and the correct classification can also be assessed using information-theoretic entropy or related measures [Dom 2001]. Several clustering engines have been evaluated following this approach (e.g., Maarek et al. [2000]; Cheng et al. [2005]; Geraci et al. [2008]). A more flexible agreement criterion that allows for justified differences between the machine-generated and the human-generated clusters has been recently proposed in Osiński and Weiss [2005] and Osiński [2006]. One example of such justified difference, for which the algorithm is not penalized, is when a larger human-identified group gets split into smaller *pure* subgroups. It should also be noted that although the external measures for assessing cluster validity are conceptually distinct from the internal measures, the former may be experimentally correlated with the latter [Stein et al. 2003].

As remarked earlier, another key aspect that needs to be evaluated is the quality of cluster labels. Aside from linguistic plausibility, the most important feature of a label is how well it describes the content of the cluster. This can be cast as a ranking problem, in which we measure the precision of the list of labels associated with the clusters assuming that the relevance of the labels has been manually assessed for each topic [Zeng et al. 2004; Spiliopoulou et al. 2005]. Alternatively, the goodness of labels can be measured by analyzing their relationships to the clusters' content, and can be expressed, for instance, in terms of informativeness [Lawrie et al. 2001].

The criteria discussed so far do not explicitly take into account the performance of the system in which the hierarchy is encompassed. As the intended use of search results clustering is to find relevant documents, it may be convenient to evaluate the hierarchies' performance on a retrieval oriented task.

[9]www.daviddlewis.com/resources/testcollections

Table VII. Comparison of Label-Driven Subtopic Reach Time of Five Clustering Algorithms on the AMBIENT Test Collection

CREDO	Lingo	Lingo3G	STC	TRSC
14.96	15.05	13.11	15.82	17.46

We now describe an experimental study of the retrieval effectiveness of the same five clustering algorithms tested for efficiency in Section 6.1.2. Our aim was to evaluate their subtopic retrieval performance in a realistic scenario in which the user actions are driven by the meaning of the cluster labels. We used the AMBIENT (AMBIgous ENTries) test collection, a dataset specifically designed for evaluating subtopic information retrieval. It consists of 44 topics, each with a set of subtopics and a list of 100 search results with corresponding subtopic relevance judgments. The topics were selected from the list of ambiguous Wikipedia entries (those with "disambiguation" in the title). The 100 search results associated with each topic were collected from Yahoo! as of January 2008, and their relevance to each of the Wikipedia subtopics was then manually assessed. The total number of retrieved subtopics (the Wikipedia subtopics for which at least one relevant search result was retrieved) is 345, with an average of 6.46 relevant results per retrieved subtopic. AMBIENT is available online[10]; for a more detailed description of the dataset see Carpineto et al. [2009].

To evaluate the subtopic retrieval performance, we used a modified version of the subtopic reach time (SRT) measure introduced in Section 7.1. More precisely, for each query's subtopic, we first manually selected the most appropriate label among the list of cluster labels created by the clustering algorithm. The SRT value was then computed by summing the number of high-level clusters and the position of the first relevant result in the cluster with the most appropriate label. For instance, the SRT for the subtopic "Mac operating system" in the Vivísimo portion of Figure 1 would be equal to 11, given by the number of clusters in the first level (10) plus the position of the first relevant search result ("Apple – Mac OS X") within the most appropriate cluster "Mac OS X" (1). When no appropriate label was found for the subtopic at hand, or the cluster associated with the most appropriate label did not contain any relevant result, we turned to the ranked list of search results (in this case the corresponding SRT was given by the number of high-level clusters plus the position of the first result relevant to the subtopic in the ranked list associated with the query). Note that in this measure, unlike the optimal SRT discussed in the preceding section, only the search results contained in a cluster with an appropriate label are considered. We refer to it as label-driven SRT. A further advantage of label-driven SRT is that the quality of cluster labels is seen as a component of the overall retrieval performance, as opposed to evaluating the quality of cluster labels per se.

We ran each of the five systems on the AMBIENT dataset, computing their label-driven SRT with the procedure described above. The results, averaged over over the set of 44 queries, are shown in Table VII. The best performance was achieved by Lingo3G, followed by CREDO, and Lingo, which are all representatives of the description-centric algorithms. The subtopic reach times found for STC (description-aware class) and TRSC (data-centric class) were higher. It is also interesting to note that the SRT value computed for the plain ranked list of search results (those produced by Yahoo!) would be equal to 22.46, thus much higher than any clustering algorithm tested in the experiment.

It is important to emphasize that the evaluation methods presented so far, including the subtopic reach time, are good at measuring specific, albeit important, aspects of the

[10]http://credo.fub.it/ambient

Table VIII. Features of Retrieval Performance Evaluation Methods

Evaluation Method	Cluster Validity	Classification Accuracy	Label Quality	Reach Time	Subtopic Reach Time	User Studies
Fully automatic	✓	✓		✓	✓	
No need for test collection	✓		✓			✓
Mathematical measures	✓	✓	✓	✓	✓	
Task oriented				✓	✓	✓
Labels handled			✓		✓	✓

overall clustering engine performance. Furthermore, they have shortcomings and may not be accepted without reservation. In fact, performance evaluation is still largely an open question. To compensate for the lack of a unified method and standard evaluation benchmarks, usually one tries to collect multiple evidence in the hope of gaining consistent results about the superiority of one method over another. Also, due to the paucity of fully reliable or comprehensive automatic evaluation methods, one common approach is to design experiments with external subjects. As discussed in the preceding section, the focus of such experiments is on modeling an interactive information-finding task and measuring relative performance [Carpineto et al. 2006, 2009; Otterbacher et al. 2006]. In addition, they may involve the use of questionnaires and surveys, by which to identify the main usability issues [Ferragina and Gulli 2005] or evaluate the quality of cluster labels [Geraci et al. 2008].

To conclude this section, in Table VIII we summarize the main characteristics of several evaluation methods.

8. FUTURE TRENDS

The cluster hierarchies built from search results are far from being perfect. Even considering a single level, it is not always easy to figure out why some clusters have been generated and others are missing. For instance, in the top level clusters created from Vivísimo in Figure 1, there are two subspecies of tiger, Siberian and White (or Bengal), but the other three subspecies (Sumatran, South China, and Indochinese) are missing. The lack of predictability of clustering output may become even more evident if the level of granularity of the generated clusters is not uniform, as is often the case. In the Vivísimo example in Figure 1, the cluster "Foundation" is on a very specific aspect of the animal meaning of tiger (a Canadian conservation organization exclusively dedicated to the preservation of wild tigers), while "Pictures" is a broad cluster containing both pictures and textual results related to various meanings of tiger.

Inconsistency is another problem. The contents of a cluster do not always correspond to the label and the navigation through the cluster subhierarchies does not necessarily lead to more specific results. For instance, the second element in the subcluster "Tiger Woods" returned from Carrot[2] in Figure 1 is an Encyclopaedia Britannica entry listing the various meanings of tiger (including "woods") and not a result on "Tiger Woods." Thus, it should be best placed at the top level of the hierarchy.

These kinds of errors are quite typical for clustering output, whereas users prefer coherent and relatively complete category systems [Hearst 2006]. If the hierarchies are not clearly understandable or predictable, the benefits of clustering may be overwhelmed by the cost of navigating through a disorderly structure, and the overall effectiveness of the system may be prejudiced [Rodden et al. 2001].

The most important research issue is thus how to improve the quality and usability of output hierarchies. Clearly, they would benefit from an improvement in the accuracy of the snippets returned by the auxiliary search engine, but here we do not address this aspect because search results are regarded as the input. By focusing on postretrieval clustering of the given search results, improvements can be made at the various stages of the whole process, as discussed in the following.

One straightforward approach is to extract more powerful features for indexing the retrieved results, although this may not automatically result in an improvement of the retrieval performance of the resulting clustering structure. The proposed techniques include using hyperlinks [Wang and Kitsuregawa 2002], named entities [Toda and Kataoka 2005], external information available on the Internet [Gabrilovich 2006], and temporal attributes [Alonso and Gertz 2006].

Another line of research is focused on generating more expressive or effective descriptions of the clusters after the construction of the cluster hierarchy. Several techniques are available. Multi-document summarization is probably the most natural way to address the expressiveness issue [Harabagiu and Lacatusu 2005], while a smoother transition from the cluster label to the cluster contents can be achieved by providing cluster previews [Osdin et al. 2002]. Finding optimal cluster representatives [Liu and Croft 2006] can be used to improve the effectiveness of cluster-based reranking, when the user is not requested to choose one of the created clusters. Clustered results might also be used as contexts for query refinement. This is suggested in Anagnostopoulos et al. [2006], where a query consisting of several weighted terms is computed for each class or cluster by using measures such as the Fisher Index [Chakrabarti et al. 1998]. Although such queries are probably not appropriate as browsing labels, they can be effectively used to retrieve further results relevant to any cluster of interest.

It is also possible to try to improve the accuracy of the hierarchy structure. As there is no clustering algorithm that is intrinsically better or more valid than the others, provided that the structure of the data set is not incompatible with the model of the algorithm [Estivill-Castro 2002], it may be convenient to combine the existing algorithms. Method combination works well for classification [Kittler et al. 1998] and query expansion [Carpineto et al. 2002], and there is evidence that this is also the case for clustering. Fred and Jain [2005] have shown that merging the results of multiple clustering helps identify clusters with non predefined shapes or structures. This ability may increase the flexibility and robustness of the method used for identifying the topics in the search results.

When the clustering process does not depend only on the search results, but is also influenced by the user characteristics, we speak of personalization. From a machine learning point of view, this is a semisupervised problem, in which the supervised knowledge is expressed by constraints that must be obeyed by the cluster structure. Typical constraints may specify document pairs that should be clustered together or rather kept separated [Wagstaff et al. 2001], or a *range tree* with admissible parameter intervals [Rigou et al. 2006], or even an order relation over the clusters [Bade and Nürnberger 2006]. Such constraints may be provided explicitly or inferred from the user behavior (e.g., from a hierarchical folder).

Instead of optimizing the construction of the hierarchy structure, one can try to reorganize a given structure based on user actions. This can be seen as another form of personalized (adaptive) clustering. The proposed techniques exploit user feedback, whether explicit or implicit (e.g., clickthrough data), to filter out parts of the hierarchy that are presumably of no interest to the user Ferragina and Gulli [2005], Haase et al. [2005]; Pierrakos and Paliouras [2005].

Besides trying to improve the basic methodology, recent research has identified new application areas of search results clustering. A particularly timely topic is the fast growing market of mobile search. The existing user interfaces, modeled after desktop plain search engines, do not adequately address the challenges posed by small-screen, limited-input, low-bandwidth devices. By contrast, search results clustering, by its very nature, has the potential of minimizing the amount of information transmitted and displayed, as well reducing tedious user actions such as scrolling and keyword entry. Such potentials have been preliminarily investigated in some recent works. Two mobile versions of CREDO, suitable for personal digital assistants and cellular phones, are described in Carpineto et al. [2006, 2009]. The systems, termed *Credino* (small CREDO, in Italian) and SmartCREDO, are exclusively based on the search results and are freely available online.[11] Other approaches such as metadata [Karlson et al. 2006] or bookmarks [De Luca and Nürnberger 2005] make use of additional knowledge.

A further challenge is represented by XML documents. In order to take advantage of both their content and structure for clustering purposes, the traditional similarity measures used by clustering algorithms have been enriched with a variety of structural features, such as tree edit distance and graph-based representation models (for a review of recent approaches see Tagarelli and Greco [2006]). XML clustering is also being investigated within INEX[12], the INitiative for the Evaluation of XML Retrieval. Coupled with the attempts at building semantic Web search engines [Ding et al. 2004], this work can pave the way toward semantic Web clustering engines.

9. CONCLUSIONS

By showing the most likely meanings for any given request, search results clustering narrows the semantic mismatching between the user need behind the request and the list of results returned by a plain search engine. The technology of Web clustering engines has reached a level of maturity in which a rich body of research has been developed and several commercial systems are being deployed.

In this article, we have presented the most important scientific and technical aspects of Web search result clustering. We have discussed the issues that must be addressed to build a Web clustering engine and have reviewed and evaluated a number of existing algorithms and systems.

A number of advances must be made before search results clustering entirely fulfills the promise of being the PageRank of the future. First, more work needs to be done to improve the quality of the cluster labels and the coherence of the cluster structure. Second, more studies on user queries must be made to understand when search results clustering is most useful. Third, there is a need for carefully engineered evaluation benchmarks to allow cross-system comparison, and to measure progress. Fourth, advanced visualization techniques might be used to provide better overviews and guide the interaction with clustered results.

ACKNOWLEDGMENTS

We are very grateful to three anonymous reviewers for their constructive suggestions and comments.

REFERENCES

ABNEY, S. 1991. Parsing by Chunks. In *Principle-Based Parsing: Computation and Psycholinguistics*, R. C. Berwick, S. P. Abney, and C. Tenny, Eds. Kluwer Academic Publishers, 257–278.

ALLEN, R. B., OBRY, P., AND LITTMAN, M. 1993. An interface for navigating clustered document sets returned by queries. In *Proceedings of the ACM Conference on Organizational Computing Systems*. ACM Press, 166–171.

[11]http://credino.dimi.uniud.it and http://smartcredo.dimi.uniud.it
[12]inex.is.informatik.uni-duisburg.de

Alonso, O. and Gertz, M. 2006. Clustering of search results using temporal attributes. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 597–598.

Anagnostopoulos, A., Broder, A., and Punera, K. 2006. Effective and efficient classification on a search-engine model. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*. ACM Press, 208–217.

Bade, K. and Nürnberger, A. 2006. Personalized hierarchical clustering. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE, 181–187.

Broder, A. 2002. A taxonomy of Web search. *ACM SIGIR Forum 36*, 2, 3–10.

Carpineto, C., Della Pietra, A., Mizzaro, S., and Romano, G. 2006. Mobile Clustering Engine. In *Proceedings of the 28th European Conference on Information Retrieval*. Lecture Notes in Computer Science, vol. 3936. Springer, 155–166.

Carpineto, C., Mizzaro, S., Romano, G., and Snidero, M. 2009. Mobile information retrieval with search results clustering: Prototypes and evaluations. *J. Amer. Soc. Inform. Sci. Tec. 60*, 5, 877–895.

Carpineto, C. and Romano, G. 2004a. *Concept Data Analysis: Theory and Applications*. Wiley.

Carpineto, C. and Romano, G. 2004b. Exploiting the potential of concept lattices for information retrieval with CREDO. *J. Univ. Comput. Sci. 10*, 8, 985–1013.

Carpineto, C., Romano, G., and Giannini, V. 2002. Improving retrieval feedback with multiple term-ranking function combinations. *ACM Trans. Inform. Syst. 20*, 3, 259–290.

Chakrabarti, S., Dom, B., Agrawal, R., and Raghavan, P. 1998. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *Inter. J. VLDB*, *7*, 3, 163–178.

Chen, H. and Dumais, S. 2000. Bringing order to the Web: Automatically categorizing search results. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, 145–152.

Cheng, D., Vempala, S., Kannan, R., and Wang, G. 2005. A divide-and-merge methodology for clustering. In *Proceedings of the 24th ACM Symposium on Principles of Database Systems*, C. Li, Ed. ACM Press, 196–205.

Cigarrán, J., Peñas, A., Gonzalo, J., and Verdejo, F. 2005. Evaluating hierarchical clustering of search results. In *Proceedings of the 12th International Conference on String Processing and Information Retrieval (SPIRE)*. Springer, 49–54.

Cole, R., Eklund, P., and Stumme, G. 2003. Document retrieval for email search and discovery using formal concept analysis. *Appl. Artif. Intell. 17*, 3, 257–280.

Cutting, D. R., Pedersen, J. O., Karger, D., and Tukey, J. W. 1992. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 318–329.

De Luca, E. W. and Nürnberger, A. 2005. Supporting information retrieval on mobile devices. In *Proceedings of the 7th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI'05)*. ACM Press, New York, NY, 347–348.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, T. K. 1990. Indexing by latent semanic analysis. *J. Amer. Soc. Inform. Sci. 41*, 6, 391–407.

Di Giacomo, E., Didimo, W., Grilli, L., and Liotta, G. 2007. Graph visualization techniques for Web clustering engines. *IEEE Trans. Visual. Comput. Graph. 13*, 2, 294–304.

Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R. S., Peng, Y., Reddivari, P., Doshi, V., and Sachs, J. 2004. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*. ACM Press, 652–659.

Dom, B. E. 2001. An information-theoretic external cluster-validity measure. Tech. rep. RJ-10219, IBM.

Dong, Z. 2002. Towards Web Information Clustering. Ph.D. thesis, Southeast University, Nanjing, China.

Eades, P. and Tamassia, R. 1989. Algorithms for drawing graphs: an annotated bibliography. Tech. rep. CS-89-90, Department of Computer Science, Brown University.

Estivill-Castro, V. 2002. Why so many clustering algorithms: A position paper. *SIGKDD Explor. 4*, 1, 65–75.

Everitt, B. S., Landau, S., and Leese, M. 2001. *Cluster Analysis*, 4th Ed. Oxford University Press.

Ferragina, P. and Gulli, A. 2004. The Anatomy of SnakeT: A Hierarchical Clustering Engine for Web-Page Snippets. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*. Lecture Notes in Computer Science, vol. 3202. Springer, 506–508.

Ferragina, P. and Gulli, A. 2005. A personalized search engine-based on Web-snippet hierarchical clustering. In *Proceedings of the 14th International Conference on World Wide Web*. ACM Press, 801–810.

FRED, A. L. N. AND JAIN, A. K. 2005. Combining multiple clusterings using evidence accumulation. *IEEE Trans. Patt. Anal. Mach. Intell. 27*, 6, 835–850.

GABRILOVICH, E. 2006. Feature generation for textual information retrieval using world knowledge. Ph.D. thesis, Technion—Israel Institute of Technology, Haifa, Israel.

GANTER, B. AND WILLE, R. 1999. *Formal Concept Analysis: Mathematical Foundations*. Springer.

GERACI, F., MAGGINI, M., PELLEGRINI, M., AND SEBASTIANI, F. 2008. Cluster generation and cluster labelling for Web snippets: A fast and accurate hierarchical solution. *Internet Math. 3*, 4, 413–443.

GIANNOTTI, F., NANNI, M., PEDRESCHI, D., AND SAMARITANI, F. 2003. WebCat: Automatic categorization of Web search results. In *Proceedings of the 11th Italian Symposium on Advanced Database Systems (SEBD)*, S. Flesca, S. Greco, D. Saccà, and E. Zumpano, Eds. Rubettino Editore, 507–518.

GREFENSTETTE, G. 1995. Comparing two language identification schemes. In *Proceedings of the 3rd International Conference on Statistical Analysis of Textual Data (JADT'95)*. 263–268.

HAASE, P., HOTHO, A., SCHMIDT-THIEME, L., AND SURE, Y. 2005. Collaborative and usage-driven evolution of personal ontologies. In *Proceedings of the 2nd European Semantic Web Conference*. Springer, 486–499.

HALKIDI, M., BATISTAKIS, Y., AND VAZIRGIANNIS, M. 2001. On clustering validation techniques. *J. Intell. Inform. Syst. 17*, 2–3, 107–145.

HARABAGIU, S. AND LACATUSU, F. 2005. Topic themes for multi-document summarization. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 202–209.

HARTIGAN, J. A. 1975. *Clustering Algorithms*. Wiley.

HEARST, M., ELLIOTT, A., ENGLISH, J., SINHA, R., SWEARINGEN, K., AND YEE, K.-P. 2002. Finding the flow in Web site search. *Comm. ACM* (Special Issue: the Consumer Side of Search) *45*, 9, 42–49.

HEARST, M. A. 2006. Clustering versus faceted categories for information exploration. *Comm. ACM 49*, 4, 59–61.

HEARST, M. A. AND PEDERSEN, J. O. 1996. Reexamining the cluster hypothesis: scatter/gather on retrieval results. In *Proceedings of the 19th ACM International Conference on Research and Development in Information Retrieval*. ACM Press, 76–84.

HERMAN, I., MELANCON, G., AND MARSHALL, S. M. 2000. Graph visualization and navigation in information visualization: A survey. *IEEE Trans. Visual. Comput. Graph. 6*, 10, 1–21.

HOTHO, A., STAAB, S., AND STUMME, G. 2003. Explaining text clustering results using semantic structures. In *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*. Lecture Notes in Computer Science, vol. 2838. Springer, 217–228.

HUSEK, D., POKORNY, J., REZANKOVA, H., AND SNASEL, V. 2006. Data clustering: From documents to the Web. In *Web Data Management Practices: Emerging Techniques and Technologies*, A. Vakali and G. Pallis, Eds. Baker and Taylor, 1–33.

JAIN, A. K. AND DUBES, R. C. 1988. *Algorithms for Clustering Data*. Prentice-Hall.

JAIN, A. K., MURTY, M. N., AND FLYNN, P. J. 1999. Data Clustering: A Review. *ACM Comput. Surv. 31*, 3, 264–323.

JOHNSON, B. AND SHNEIDERMAN, B. 1991. Treemaps: A space-filling approach to the visualization of hierarchical information structures. In *Proceedings of IEEE Visualization*. IEEE Computer Society, San Diego, 284–291.

KÄKI, M. 2005. Findex: Search result categories help users when document ranking fails. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'05)*. ACM Press, 131–140.

KANTROWITZ, M., MOHIT, B., AND MITTAL, V. 2000. Stemming and its effects on TFIDF ranking. In *Proceedings of the 23rd ACM International Conference on Research and Development in Information Retrieval*. ACM Press, 357–359.

KARLSON, A. K., ROBERTSON, G. G., ROBBINS, D. C., CZERWINSKI, M., AND SMITH, G. 2006. FaThumb: A facet-based interface for mobile search. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM Press, 711–720.

KATIFORI, A., HALATSIS, C., LEPOURAS, G., VASSILAKIS, C., AND GIANNOPOULOU, E. 2007. Ontology visualization methods a survey. *ACM Comput. Surv. 39*, 4, 1–43.

KITTLER, J., HATEF, M., DUIN, R. P., AND MATAS, J. 1998. On Combining Classifiers. *IEEE Trans. Patt. Anal. Mach. Intell. 20*, 3, 226–239.

KUMMAMURU, K., LOTLIKAR, R., ROY, S., SINGAL, K., AND KRISHNAPURAM, R. 2004. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *Proceedings of the 13th International Conference on World Wide Web*. ACM Press, 658–665.

LAWRIE, D. J. AND CROFT, B. W. 2003. Generating hiearchical summaries for Web searches. In *Proceedings of the 26th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 457–458.

LAWRIE, D. J., CROFT, B. W., AND ROSENBERG, A. 2001. Finding topic words for hierarchical summarization. In *Proceedings of the 24th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 349–357.

LEUSKI, A. 2001. Evaluating document clustering for interactive information retrieval. In *Proceedings of the 10th International Conference on Information and Knowledge Management*. ACM Press, 33–40.

LEUSKI, A. AND CROFT, B. W. 1996. An evaluation of techniques for clustering search results. Tech. rep. IR-76, University of Massachusetts, Amherst.

LIN, D. AND PANTEL, P. 2002. Concept discovery from text. In *Proceedings of the 19th International Conference on Computational Linguistics*. Association for Computational Linguistics, 1–7.

LIU, T., LIU, S., CHEN, Z., AND MA, W.-Y. 2003. An evaluation on feature selection for text clustering. In *Proceedings of the 20th International Conference on Machine Learning, August 21–24*, T. Fawcett and N. Mishra, Eds. AAAI Press, 488–495.

LIU, X. AND CROFT, B. W. 2004. Cluster-based retrieval using language models. In *Proceedings of the 27th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 186–193.

LIU, X. AND CROFT, B. W. 2006. Representing clusters for retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 671–672.

MAAREK, Y. S., FAGIN, R., BEN-SHAUL, I. Z., AND PELLEG, D. 2000. Ephemeral document clustering for Web applications. Tech. rep. RJ 10186, IBM Research.

MANBER, U. AND MYERS, G. 1993. Suffix Arrays: A new method for on-line string searches. *SIAM J. Comput. 22*, 5, 935–948.

MANNING, C. D., RAGHAVAN, P., AND SCHÜTZE, H. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

MANNING, C. D. AND SCHÜTZE, H. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

MASLOWSKA, I. 2003. Phrase-based hierarchical clustering of Web search results. In *Proceedings of the 25th European Conference on IR Research, (ECIR)*. Lecture Notes in Computer Science, vol. 2633. Springer, 555–562.

MENG, W., YU, C., AND LIU, K.-L. 2002. Building efficient and effective metasearch engines. *ACM Comput. Surv. 34*, 1, 48–89.

NGO, C. L. AND NGUYEN, H. S. 2004. A tolerance rough set approach to clustering Web search results. In *Proceedings of the Knowledge Discovery in Databases: PKDD*. Lecture Notes in Computer Science, vol. 3202. Springer, 515–517.

OSDIN, R., OUNIS, I., AND WHITE, R. W. 2002. Using hierarchical clustering and summarisation approaches for Web retrieval. In *Proceedings of the 11th Text REtrieval Conference (TREC)*. National Institute of Standards and Technology (NIST).

OSIŃSKI, S. 2006. Improving quality of search results clustering with approximate matrix factorisations. In *Proceedings of the 28th European Conference on Information Retrieval*. Lecture Notes in Computer Science, vol. 3936. Springer, 167–178.

OSIŃSKI, S., STEFANOWSKI, J., AND WEISS, D. 2004. Lingo: Search results clustering algorithm based on singular value decomposition. In *Proceedings of the International Intelligent Information Processing and Web Mining Conference*. Advances in Soft Computing. Springer, 359–368.

OSIŃSKI, S. AND WEISS, D. 2005. A concept-driven algorithm for clustering search results. *IEEE Intell. Syst. 20*, 3, 48–54.

OTTERBACHER, J., RADEV, D. R., AND KAREEM, O. 2006. News to go: hierarchical text summarization for mobile devices. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 589–596.

PANTEL, P. AND LIN, D. 2002. Document Clustering With Committees. In *Proceedings of the 25th ACM International Conference on Research and Development in Information Retrieval*. ACM Press, 199–206.

PERUGINI, S. AND RAMAKRISHNAN, N. 2006. Interacting with Web hierarchies. *IT Profess. 8*, 4, 19–28.

PIERRAKOS, D. AND PALIOURAS, G. 2005. Exploiting probabilistic latent information for the construction of community Web directories. In *Proceedings of the 10th International Conference on User Modeling*. Springer, 89–98.

PORTER, M. F. 1997. An algorithm for suffix stripping. In *Readings in Information Retrieval*, K. S. Jones and P. Willett, Eds. Morgan Kaufmann, 313–316.

RIGOU, M., SIRMAKESSIS, S., AND TZIMAS, G. 2006. A method for personalized clustering in data intensive Web applications. In *Proceedings of the Joint International Workshop on Adaptivity, Personalization and the Semantic Web, (APS)*. ACM Press, 35–40.

RIVADENEIRA, W. AND BEDERSON, B. B. 2003. A study of search result clustering interfaces: Comparing textual and zoomable user interfaces. Tech. rep. HCIL-TR-2003-36, University of Maryland.

ROBERTS, J. C. 1998. On encouraging multiple views for visualization. In *Proceedings of IEEE Symposium on InfoVis*. IEEE Computer Society, 8–14.

RODDEN, K., BASALAJ, W., SINCLAIR, D., AND WOOD, K. R. 2001. Does organisation by similarity assist image browsing? In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM Press, 190–197.

ROSE, D. E. AND LEVINSON, D. 2004. Understanding user goals in Web search. In *Proceedings of the 13th International Conference on World Wide Web*. ACM Press, 13–19.

SAHOO, N., CALLAN, J., KRISHNAN, R., DUNCAN, G., AND PADMAN, R. 2006. Incremental hierarchical clustering of text documents. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*. 357–366.

SALTON, G., WONG, A., AND YANG, C. S. 1975. A vector space model for automatic indexing. *Comm. ACM 18*, 11, 613–620.

SARKAR, M. AND BROWN, M. 1994. Graphical fisheye views. *Comm. ACM 37*, 12, 73–84.

SCHÜTZE, H. AND SILVERSTEIN, C. 1997. Projections for efficient document clustering. In *Proceedings of the 20th ACM International Conference on Research and Development in Information Retrieval*. ACM Press, 74–81.

SEBASTIANI, F. 2002. Machine learning in automated text categorization. *ACM Comput. Surv. 34*, 1, 1–47.

SPILIOPOULOU, M., SCHAAL, M., MÜLLER, R. M., AND BRUNZEL, M. 2005. Evaluation of ontology enhancement tools. In *Proceedings of the Semantics, Web and Mining, Joint International Workshops, EWMF and KDO*. Lecture Notes in Computer Science, vol. 4289. Springer, 132–146.

STASKO, J. AND ZHANG, E. 2000. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Proceedings of IEEE Symposium on InfoVis*. IEEE Computer Society, 57–65.

STEFANOWSKI, J. AND WEISS, D. 2003a. Carrot^2 and language properties in Web search results clustering. In *Proceedings of the 1st International Atlantic Web Intelligence Conference*. Lecture Notes in Computer Science, vol. 2663. Springer, 240–249.

STEFANOWSKI, J. AND WEISS, D. 2003b. Web search results clustering in Polish: Experimental Evaluation of Carrot. In *Proceedings of the International Intelligent Information Processing and Web Mining Conference*. Advances in Soft Computing. Springer, 209–218.

STEIN, B. AND MEYER ZU EISSEN, S. 2004. Topic identification: Framework and application. In *Proceedings of the 4th International Conference on Knowledge Management*. 353–360.

STEIN, B., MEYER ZU EISSEN, S., AND WIBROCK, F. 2003. On cluster validity and the information need of users. In *Proceedings of the 3rd IASTED International Conference on Artificial Intelligence and Applications (AIA)*. Springer, 216–221.

TAGARELLI, A. AND GRECO, S. 2006. Toward semantic XML clustering. In *Proceedings of the 6th SIAM International Conference on Data Mining (SDM)*. 188–199.

TEEVAN, J., ALVARADO, C., ACKERMAN, M. S., AND KARGER, D. K. 2004. The perfect search engine is not enough: a study of orienteering behavior in directed search. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM Press, 415–422.

TODA, H. AND KATAOKA, R. 2005. A search result clustering method using informatively named entities. In *Proceedings of the 7th ACM International Workshop on Web Information and Data Management (WIDM)*. ACM Press, 81–86.

TOMBROS, A., VILLA, R., AND VAN RIJSBERGEN, K. 2002. The effectiveness of query-specific hierarchic clustering in information retrieval. *Inform. Proc. Manag. 38*, 4, 559–582.

TURETKEN, O. AND SHARDA, R. 2005. Clustering-based visual interfaces for presentation of Web search results: An empirical investigation. *Inform. Syst. Front. 7*, 3, 273–297.

UKKONEN, E. 1995. On-line construction of suffix trees. *Algorithmica 14*, 3, 249–260.

VAN RIJSBERGEN, K. 1979. *Information Retrieval*. Butterworth-Heinemann.

WAGSTAFF, K., CARDIE, C., ROGERS, S., AND SCRÖDL, S. 2001. Constrained K-means clustering with background knowledge. In *Proceedings of the 18th International Conference on Machine Learning*. Morgan Kaufmann, 577–584.

Wang, X. and Zhai, C. 2007. Learn from Web search logs to organize search results. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 87–94.

Wang, Y. and Kitsuregawa, M. 2002. On combining link and contents information for Web page clustering. In *Proceedings of the 13th International Conference on Database and Expert Systems Applications (DEXA)*. Springer, 902–913.

Weiss, D. 2006. Descriptive clustering as a method for exploring text collections. Ph.D. thesis, Poznan University of Technology, Poznań, Poland.

Willet, P. 1988. Recent trends in hierarchic document clustering: A critical review. *Inform. Proc. Manag. 24*, 5, 577–597.

Yang, Y. and Pedersen, J. O. 1997. A comparative study on feature selection in text categorization. In *Proceedings of the 14th International Conference on Machine Learning (ICMC)*. Morgan Kaufmann, San Francisco, 412–420.

Ye, S., Chua, T.-S., and Kei, J. R. 2003. Querying and clustering Web pages about persons and organizations. In *Proceedings of the IEEE/WIC International Conference on Web Intelligence*. Springer, 344–350.

Zamir, O. and Etzioni, O. 1998. Web document clustering: A feasibility demonstration. In *Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 46–54.

Zamir, O. and Etzioni, O. 1999. Grouper: A dynamic clustering interface to Web search results. *Comput. Netw. 31*, 11–16, 1361–1374.

Zeng, H.-J., He, Q.-C., Chen, Z., Ma, W.-Y., and Ma, J. 2004. Learning to cluster Web search results. In *Proceedings of the 27th ACM International Conference on Research and Development in Information Retrieval*. ACM Press, 210–217.

Zhang, D. and Dong, Y. 2004. Semantic, hierarchical, online clustering of Web search results. In *Proceedings of 6th Asia-Pacific Web Conference (APWeb)*. Lecture Notes in Computer Science, vol. 3007. Springer, 69–78.

Zhang, Y.-J. and Liu, Z.-Q. 2004. Refining Web search engine results using incremental clustering. *Int. J. Intell. Syst. 19*, 191–199.

Zhao, H., Meng, W., Wu, Z., Raghavan, V., and Yu, C. 2005. Fully automatic wrapper generation for search engines. In *Proceedings of the 14th International Conference on World Wide Web*. ACM Press, 66–75.

Received December 2007; revised May 2008; accepted August 2008