Semantic, Hierarchical, Online Clustering of Web Search Results

Dell Zhang[1,2]
[1] Department of Computer Science
School of Computing
S15-05-24, 3 Science Drive 2
National University of Singapore
Singapore 117543
[2] Singapore-MIT Alliance
E4-04-10, 4 Engineering Drive 3
Singapore 117576
+65-68744251
dell.z@ieee.org

Yisheng Dong
Department of Computer Science & Engineering
Southeast University
Nanjing, 210096, P.R.China
+86-25-3319900
ysdong@seu.edu.cn

ABSTRACT

Today, search engine is the most commonly used tool for Web information retrieval, however, its current status is still far from satisfaction. This paper focuses on clustering Web search results in order to help users find relevant Web information more easily and quickly. The main contributions of this paper include the following. (1) The benefits of using key phrases as natural language information features are discussed. An effective and efficient algorithm based on suffix array for key phrase discovery is presented. The efficiency of this method is very high no matter how large the language's alphabet is. (2) The concept of orthogonal clustering is proposed for general clustering problems. The reason why matrix SVD (Singular Value Decomposition) can provide solution to orthogonal clustering is strictly proved. The orthogonal clustering algorithm has a solid mathematics foundation and many advantages over traditional heuristic clustering algorithms. (3) The WICE system is designed and implemented to automatically organize multilingual Web search results through a semantic, hierarchical, online clustering approach named SHOC.

Keywords

Web information retrieval, clustering.

1. INTRODUCTION

With information explosion on the Web as well as popularity of Internet, the Web has already become the biggest data source for various applications. However, how to obtain high-quality information from the Web effectively and efficiently according to user's query request has created big challenges for many computer science disciplines, such as information retrieval and data mining, because of the features of the Web (huge volume, heterogeneous, dynamic, semi-structured, etc.).

Web search engine is the most commonly used tool for information retrieval on the web; however, its current status is far from satisfaction for several possible reasons [10]:

- Very limited coverage of the Web;
- Outdated pages and broken hyperlinks;
- Many useful or relevant pages are not returned
- Many returned pages are useless or irrelevant;
- Users may be just interested in " more qualified" information or small part of information returned while thousands of pages are returned from search engine;
- Different users have different requirements and expectations for search results;
- Sometimes search requests can not be expressed clearly just in several keywords;
- The phenomena of synonymy (several words may correspond to same concept) and polysemy (one word may have several different meanings) make things more complicated;
- .....

While efforts to improve Web crawling, searching and ranking may alleviate the first five problems, further studies are needed for the others. We think that clustering of Web search results could help a lot. If Web search results are presented in groups, users will be able to have an overview of the whole topic or just select interested groups to browse.

We propose to automatically group Web search results through a semantic, hierarchical, online clustering approach named SHOC, which is an extension of O. Zamir and O. Etzioni's work [12]. By combining the power of two novel techniques, key phrase discovery and orthogonal clustering, SHOC can generate both reasonable and readable clusters. Moreover, SHOC can work for multiple languages: not only English but also oriental languages like Chinese.

This paper is organized as follows. In section 2, we briefly review related work. In section 3, we present the proposed approach SHOC. In section 4, we describe the prototype system WICE. In section 5, we make conclusions.

2. RELATED WORK

D. Cutting et al. have created the Scatter/Gather system to cluster Web search results [3]. However, their system has some limitations due to the shortcomings of the traditional heuristic clustering algorithms (e.g. k-means) they used.

Y. Wang et al. have proposed an interesting method to cluster Web search results based on hyperlinks [10]. But their method needs to download and parse the original Web pages, so it is not able to provide clustering results quickly.

Our approach to cluster Web search results is an extension of O. Zamir and O. Etzioni's work [12]. They proposed in [12] an algorithm named Suffix Tree Clustering (STC) to automatically group Web search results. STC first constructs a suffix tree where each internal node corresponds to a phrase, then form base clusters by grouping the Web search results that contain same "key" phrase, finally merge base clusters that have large overlap.

STC has made substantial progress on clustering of Web search results, but still has several drawbacks:

- its key phrase discovery algorithm based on suffix tree is inappropriate for oriental languages like Chinese;
- documents containing no key phrase become inaccessible since they are not included in any cluster;
- taking the hierarchy of the constructed suffix tree directly as the hierarchy of the generated clusters is not reasonable,
- the phenomena of synonymy and polysemy are neglected.

Our work attempts to overcome these shortcomings of STC.

3. SHOC APPROACH

A practical clustering approach for Web search results should meet the following requirements.

- Semantic. The clustering algorithm should group search results based on their semantic topic. Since a search result may have multiple topics, it is instructive not to confine one search result in only one cluster. The clustering algorithm should also provide each cluster a label that describes the cluster topic, so that users can determine at a glance whether a cluster is of his/her interest.
- Hierarchical. The clustering algorithm should automatically organize the generated clusters into a tree structure to facilitate user browsing.
- Online. The clustering algorithm should be able to provide fresh clustering results "just-in-time", because for the impatient users each second counts.

These requirements will be emphasized throughout this paper.

We propose to automatically group Web search results through a Semantic, Hierarchical, Online Clustering (SHOC) approach, which is composed of three major steps: (1) data collection and cleaning; (2) feature extraction; (3) identifying and organizing clusters.

3.1 Data Collection and Cleaning

The data collection task here is actually meta-search. Given a query, we just forward it to several search engines and then collect their search results (lists of pointers to Web pages). Usually the search results returned by a search engine are partitioned into several result-pages to facilitate user browsing. To get high efficiency for meta-search, we use a two-level parallelization mechanism: (1) search engines are called in parallel through multi-threads; (2) a search engine's all result-pages are fetched in parallel through multi-threads. Then we synthesize all the lists of search results from different search engines into a uniform ranked list of search results. For duplicate search results, only the one with highest rank will be reserved.

A search result usually includes the URL, title and snippet of its corresponding Web page. Figure 1 illustrates a search result returned by Google (http://www.google.com/) for a Chinese query "**数据挖掘**" (data mining). Note the " online" requirement" implies that we do not have time to download the original complete Web pages that the search results point to. In reality, most users are unwilling to wait for the clustering system to download the original Web pages. Therefore we take a Web page's title and snippet in the search results as a good summary of its content, and use it as a "document" to be fed to the clustering algorithm.

Figure 1, a search result returned by Google for a Chinese query "数据挖掘" (data mining).

Each document is parsed and split into sentences according to punctuations (period, comma, semicolon, question mark etc.) and HTML tags (<p>,
, <li>, <td> etc.). The non-word tokens are stripped and redundant spaces are compressed. The English words are stemmed using a stemming algorithm. For instance, the search result shown in Figure 1 is transformed into the document shown in Figure 2.

产品
IBM ***推出新型数据挖掘技术***
IBM ***近日发布了一项新型的基于标准的数据挖掘技术***
IBM DB2 ***智能挖掘器积分服务***
IBM DB2 Intelligent Miner Scoring Service
它可以帮助企业轻松地为自己的客户和供应商开发出个性化的解决方案

Figure 2, the document corresponding to the sample search result shown in Figure 1.

3.2 Feature Extraction

Most clustering algorithms treat a document as a " bag-of-words", totally ignoring word order and proximity which may provide important information for clustering. In contrast, we decide to use key phrases extracted of the document collection as document features. The benefit is two-fold: (1) it can improve the quality of the clusters through leveraging more information present in the documents; (2) it is helpful to construct concise and accurate labels for the generated clusters [12]. The key phrase discovery algorithm for the document collection is the same as that for single document, because a document collection can be treated as a pseudo-document.

In our scenario, a document is essentially a string of characters, and a key phrase is defined as a meaningful substring within a sentence which is both specific and significant. The reason every phrase is restricted to be within a sentence is because sentence boundaries usually indicate some topical shift. This also reduces the cost of our key phrase discovery algorithm.

Given a document *T* of length *N*, we check if a substring *S* of *T* can be a key phrase through the three measures: completeness, stability, and significance.

DEFINITION Suppose *S* occurs in *k* distinct positions $p_1, p_2, \ldots, p_k$ in *T*, *S* is " complete" if and only if the $(p_i\text{-}1)$th token in *T* is different with the $(p_j\text{-}1)$th token for at least one (i, j) pair, $1 \le i < j \le k$ (called "left-complete"), and the $(p_i+|S|)$th token is different with the $(p_j+|S|)$th token for at least one (i, j) pair, $1 \le i < j \le k$ (called "right-complete"). [14]

DEFINITION Suppose $S = "c_1 c_2 \cdots c_P"$, the stability (mutual information) of S is $MI(S) = \dfrac{f(S)}{f(S_L) + f(S_R) - f(S)}$, where $S_L = "c_1 \cdots c_{P-1}"$, $S_R = "c_2 \cdots c_P"$, and $f(S), f(S_L), f(S_R)$ are frequencies of S, S_L, S_R.

DEFINITION The significance of S can be estimated as $se(S) = f(S) \times g(|S|)$, where $f(S)$ is the frequency of S, $|S|$ is the length of S, $g(x)$ is a heuristic utility function the string length, $g(1) = 0$, $g(x) = \log_2 x$ when $2 \le x \le 8$ and $g(x) = 3$ when $x > 8$.

The task of key phrase discovery in English could be accomplished efficiently using the suffix tree data structure [6], as described in [12]. However, the performance (time and space complexity) of suffix tree is related to the alphabet size of language [9]. As we all know, oriental languages have much larger alphabet that English, e.g., Chinese has more than 6,000 characters, hereby using suffix tree is not efficient for key phrase discovery in oriental language texts.

Besides, oriental languages like Chinese do not have explicit word separators (such as blanks in English) [2]. Therefore straightforwardly applying English key phrase discovery algorithms to Chinese may extract many meaningless partial phrases, e.g., "**数据挖**" (part of the real phrase "**数据挖掘**").

Here we present a novel key phrase discovery algorithm based on suffix array, which is both scalable over alphabet size and able to avoid extracting meaningless partial phrases.

The suffix array data structure was introduced as a text indexing technique [9]. Using a suffix array, online string searches of the type, " Is *S* a substring of *T*?" can be answered in O(*P*+log*N*) time, where *P* is the length of *S* and *N* is the length of *T*. Such efficiency is competitive with (and in some cases slightly better than) that of using a suffix tree. A suffix array can be constructed with O(*N*) expected time complexity, regardless of the alphabet size. The major advantage of using suffix array over using suffix tree is in space. A suffix tree needs O(*N* |S|) space that grows with alphabet size |S|. Manber and Myers [9] reported that suffix arrays are an order of magnitude more efficient in space than suffix trees even in the case of relatively small alphabet size (|S| = 96). The advantage of using suffix array over using suffix trees is significant for large alphabet languages like Chinese.

The suffix array *s* of a document *T*, is an array of all *N* suffixes of *T*, sorted alphabetically. A suffix (also known as semi-infinite string) *s*[*i*], is a string that starts at position *i* in the text and continues to the end of the text. In practice, a suffix *s*[*i*] is typically denoted by a 4-byte integer, *i*, which one might have thought would require O(*N*) space. Manber and Myers' algorithm [9] uses a LCP array to accelerate searching operation. A LCP array *lcp* stores *N*+1 integer elements, where *lcp*[*i*] $(1 \le i \le N-1)$ indicates the length of the longest common prefix between s[*i*-1] and *s*[*i*], *lcp*[0]=*lcp*[*N*]=0. The LCP array can also be constructed with O(*N*) expected time complexity. For example, consider a simple document "**to_be_or_not_to_be**", its suffix array *s* and LCP array *lcp* is shown in Figure 3.

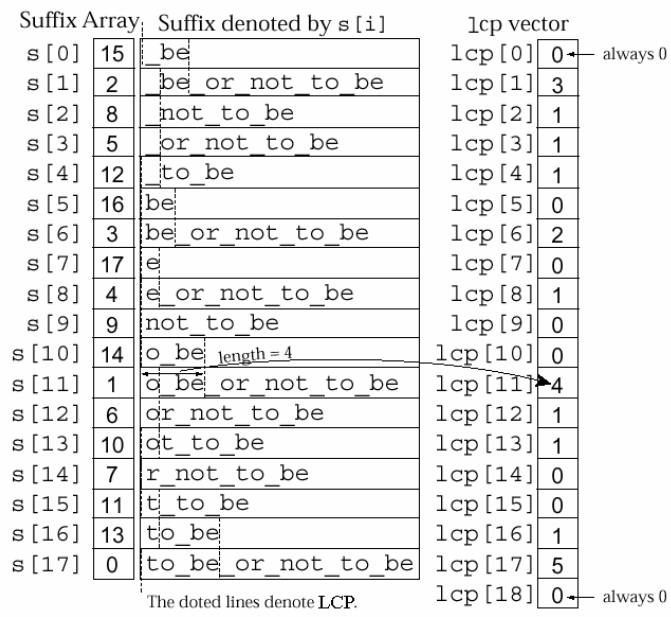

Figure 3 (adopted from 11), the suffix array *s* and its LCP array *lcp* of "to_be_or_not_to_be".

Given a document *T* of length *N*, a set of key phrases can be efficiently extracted, taking advantage of its suffix array *s* and its LCP array *lcp*.

THEOREM A substring *S* of *T* is right-complete if and only if there is a *w* ($1 \le w < N$) and *S* is the LCP of $s[w-1]$ and $s[w]$.

PROOF Omitted due to space limit.

It turns out that every right-complete substring (including complete substring) of *T*, can be identified by the position of a pair of adjacent suffixes in the suffix array *s*.

DEFINITION Assume *RCS* is a right-complete substring of *T*, the identification code of *RCS* is

$$ID(RCS) = \min\{\, w \mid 1 \le w < N \text{, the LCP of } s[w-1] \text{ and } s[w] \text{ is } RCS \,\}$$

There are at most *N-1* right-complete substrings, even though there are $N(N+1)/2$ substrings of *T*.

Based on the above theorem, we propose a linear time complexity algorithm **discover_rcs** (shown in Figure 4), to extract all right-complete substrings of *T* and meanwhile count their frequencies. The **discover_rcs** algorithm leverages a stack to store the right-complete substrings under counting. Figure 5 depicts three cases that may be encountered while scanning *lcp*, where *RCS** represents the current right-complete substring on the stack top. It is obvious that the time complexity of the **discover_rcs** algorithm is O(*N*).

```
void discover_rcs ( )
{
    typedef structure {
        int ID;
        int frequency;
    } RCSTYPE;
    RCSTYPE rcs_stack[N];    // N is the document's length
    Initialize rcs_stack;
    int sp = -1;   // the stack pointer
    int i = 1;
    while ( i < N+1 ) do {
        if ( sp < 0 ) {   // the stack is empty
            if ( lcp[i] > 0 ) {
                sp ++;
                rcs_stack[sp].ID = i;
                rcs_stack[sp].frequency = 2;
            }
            i ++;
        }
        else {
            int r = rcs_stack[sp].ID;
            if ( lcp[r] < lcp[i]) {   // case (a)
                sp ++ ;
                rcs_stack[sp]. ID = i ;
                rcs_stack[sp]. frequency = 2 ;
                i ++ ;
            }
            else if ( lcp[r] == lcp[i] ) {   // case (b)
                rcs_stack[sp].frequency ++ ;
                i ++;
            }
            else {   // case (c)
                Output rcs_stack[sp];     // ID & frequency
                int f = rcs_stack[sp].frequency;
                sp - -;
                if ( sp >= 0 ) {
                    rcs_stack[sp].frequency = rcs_stack[sp].frequency + f - 1;
                }
            }   // end of case (c)
        }   // end of if (sp < 0)
    }   // end of while
}
```

Figure 4, the discover_rcs algorithm.

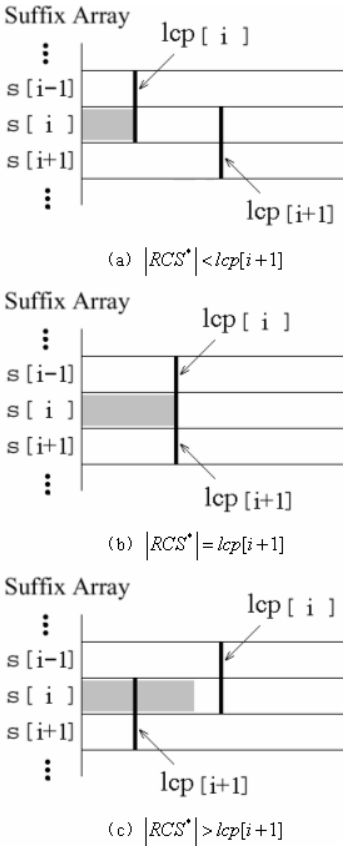

Figure 5, the 3 cases may be encountered while scanning *lcp*.

The output of the **discover_rcs** algorithm for the sample document is showed in Figure 6.

rcs array

ID	frequency	*RCS*
1	2	_be
2	5	_
6	2	be
8	2	e
11	2	o_be
12	4	o
17	2	to_be
16	3	t

Figure 6, the output of the discover_rcs algorithm for the sample document.

A complete substring should be both right-complete and left-complete. To discover all the left-complete substrings, we just apply the **discover_rcs** algorithm to ~*T*, the inverse document *T*. If *S* is a right-complete substring of ~*T*, then ~*S* must be a left-complete substring of *T*. The inverse sample document and its suffixes are showed in Figure 7.

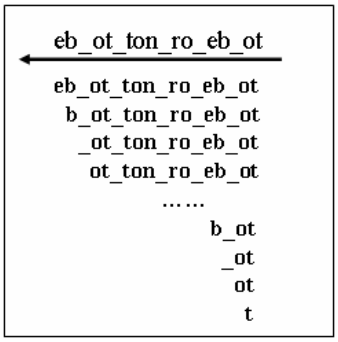

Figure 7, the inverse sample document and its suffixes.

Suppose *rcs* is the array of right-complete substrings, and *lcs* is the array of left-complete substrings. The array *rcs* is already alphabetically ordered. We also sort the array *lcs* to make it alphabetically ordered. Then we use the **intersect_lcs_rcs** algorithm (shown in Figure 8) to find the intersection of *lcs* and *rcs*. The output of this algorithm is the alphabetically ordered array of the complete substrings of *T* with their frequencies Suppose the size of *lcs* and *rcs* are *L* and *R* respectively, the time complexity of the **intersect_lcs_rcs** algorithm is O(*L*+*R*).

```
void intersect_lcs_rcs ( sorted lcs array, sorted rcs array )
{
    int i = 0;
    int j = 0;
    while ( ( i < L ) && ( j < R ) ) {
        string str_l = lcs[i].ID denoted left-complete substring;
        string str_r = rcs[j].ID denoted right-complete substring ;
        if ( str_l == str_r ) {
            Output lcs[i];
            i ++;
            j ++;
        }
        if ( str_l < str_r ) {
            i ++;
        }
        if ( str_l > str_r ) {
            j ++;
        }
    }
}
```

Figure 8, the intersect_lcs_rcs algorithm.

The output of the **interect_lcs_rcs** algorithm for the sample document is showed in Figure 9.

cs array

ID	frequency	*CS*
2	5	_
12	4	o
16	3	t
17	2	to_be

Figure 9, the output of the intersect_lcs_rcs algorithm for the sample document.

After obtaining the alphabetically ordered array of the complete substrings of *T* with their frequencies, we can get the frequency of any complete substring using binary search algorithm. Thereafter, the stability (mutual information) and significance estimation of each complete string could be computed easily.

To discover key phrases from the set of phrases (complete substrings), we only need to examine every complete substring to see whether or not it is stable and significant. The complete, stable, significant substrings are just the key phrases we need.

M. Yamamoto and K. W. Church have developed an algorithm using suffix array to compute term frequency and document frequency for all substrings in a document of size *N*, in O(*N*log*N*) time [11]. Their algorithm seems to be more complex than ours, as we only consider " complete" substrings. L.F. Chien has proposed more strict conditions of string " completeness" [13], LCD (Left Context Dependency) and RCD (Right Context Dependency), which is computationally more expensive.

3.3 Identifying and Organizing Clusters

The STC algorithm simply groups documents sharing a common phrase into one cluster [12]. However, such lexical methods can be inaccurate and incomplete. Since there are usually many ways to express a given concept, the literal terms may not match those of a relevant document. In addition, most words have multiple meanings, so the terms will literally match terms in irrelevant documents. A better clustering approach should run on semantic level, i.e., group documents sharing a common conceptual topic together.

It is natural to assume that a document belongs to a cluster in some degree. So we adopt a continuous cluster definition here.

DEFINITION A cluster of *m* objects $\mathbf{t}_1, \mathbf{t}_2, \cdots, \mathbf{t}_m$, C_q, can be identified by a *m*-dimensional vector $\mathbf{x}_q$, $|\mathbf{x}_g| = 1$ and $\mathbf{x}_g(i)$ represents the degree in which $\mathbf{t}_i$ belongs to C_q. $\mathbf{x}_q$ is denoted as the cluster vector of C_q, and it can be used interchangeably with C_q.

After the previous steps, we can get *m* key phrases of *n* documents. Taking key phrases as terms, the search results can be described as a *m*×*n* term-document matrix $\mathbf{A}$, whose row vectors represent the terms and column vectors represent the documents. The element $\mathbf{A}(i, j) = 1$ if the *i*-th term *Ti* occurs in the *j*-th document *Dj*, or $\mathbf{A}(i, j) = 0$.

Following the idea of idea of Latent Semantic Indexing (LSI) [1][4], we attempt to discover the latent semantic of documents through analyzing the associations between terms and documents. The term-document matrix $\mathbf{A}$ could be visualized as a bipartite graph, as shown in Figure 10. The bipartite graph reveals the dual reinforcing relationship between terms and documents, i.e., the terms linked with the same document should be close in semantic space, and the documents linked with the same terms should be close in semantic space. That is to say, densely-linked terms or documents are close to each other in semantic space, so they should be grouped together to form a cluster.

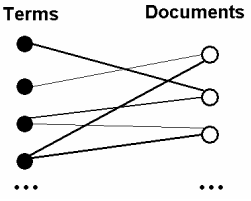

Figure 10, the associations between terms and documents.

The degree of associations among objects in a cluster can be measured by the following notation.

DEFINITION Suppose $\mathbf{x}_q$ ($\mathbf{y}_q$) is a cluster of the row (column) vectors of $\mathbf{A}$, then the cluster density of $\mathbf{x}_q$ ($\mathbf{y}_q$) is $|\mathbf{x}_g{}^T\mathbf{A}|$ ($|\mathbf{A}\mathbf{y}_q|$). [15]

We want to find the clusters with high densities since they capture main topics of the documents. Suppose $\mathbf{x}_1$ is the cluster with maximum density, and $\mathbf{x}_2$ is another cluster. It is known from basic linear algebra that $\mathbf{x}_2$ can be written as $\mathbf{x}_2 = h\mathbf{x}_1 + (\sqrt{1-h^2})\mathbf{z}$, where h is a constant scalar ($0 \le h \le 1$), $\mathbf{z} \perp \mathbf{x}_1$ and $|\mathbf{z}| = 1$. Then the cluster density of $\mathbf{x}_2$ is $|\mathbf{x}_2{}^T\mathbf{A}| = \sqrt{h^2|\mathbf{x}_1{}^T\mathbf{A}|^2 + (1-h^2)|\mathbf{z}^T\mathbf{A}|^2}$. The larger the value of h, the higher the cluster density of $\mathbf{x}_2$. If there is no constraint on $\mathbf{x}_2$, it will be arbitrary close to $\mathbf{x}_1$. To get a new meaningful cluster $\mathbf{x}_q$, we should restrict $\mathbf{x}_q$ orthogonal to the already discovered cluster vectors [15]. This leads to the following definition.

DEFINITION The orthogonal clustering of row (column) vectors of $\mathbf{A}$ is discovering a set of cluster vectors $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k$, $\mathbf{x}_q$ $(1 \le g \le k)$ is the cluster with maximum density subject to being orthogonal to $\mathbf{x}_1, \cdots, \mathbf{x}_{q-1}$.

To find out the solution for the orthogonal clustering problem, we introduce the following definition and theorem.

DEFINITION Suppose $\mathbf{M}$ is a real a *m*×*m* symmetrical matrix, the Rayleigh Quotient of $\mathbf{M}$ w.r.t $\mathbf{x} \in R^m$ is

$$R(\mathbf{x}) = \frac{\mathbf{x}^T \mathbf{M} \mathbf{x}}{\mathbf{x}^T \mathbf{x}} .$$

THEOREM Suppose $\mathbf{M}$ is a real a $m \times m$ symmetrical matrix, its eigenvalues are $l_1 \geq l_2 \geq \cdots \geq l_m$ corresponding to orthonormal eigenvectors $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m$,

if $\mathbf{x} \in R^m$, then

$$\max_{\mathbf{x} \neq \mathbf{0}} R(\mathbf{x}) = R(\mathbf{p}_1) = l_1, \ \min_{\mathbf{x} \neq \mathbf{0}} R(\mathbf{x}) = R(\mathbf{p}_m) = l_m;$$

if $\mathbf{x} \in L(\mathbf{p}_q, \mathbf{p}_{q+1}, \cdots, \mathbf{p}_h)$, $1 \leq g \leq h \leq m$, then

$$\max_{\mathbf{x} \neq \mathbf{0}} R(\mathbf{x}) = R(\mathbf{p}_q) = l_q, \ \min_{\mathbf{x} \neq \mathbf{0}} R(\mathbf{x}) = R(\mathbf{p}_h) = l_h .$$

DEFINITION Suppose a $m \times n$ matrix $\mathbf{A}$ with $rank(\mathbf{A}) = r$, $l_1 \geq l_2 \geq \cdots \geq l_r > 0$ are r non-zero eigenvalues of $\mathbf{A}\mathbf{A}^T$ ($\mathbf{A}^T\mathbf{A}$), $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$ ($\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n$) are the corresponding orthonormal eigenvectors, then the SVD (Singular Value Decomposition) of $\mathbf{A}$ is defined as $\mathbf{A} = \mathbf{U} \begin{pmatrix} \Sigma & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{pmatrix} \mathbf{V}^T$, where $\Sigma = diag(s_1, \ldots\ldots, s_r)$, $s_q = \sqrt{l_q}$ $(g = 1, 2, \cdots, k)$ are called the singular values of $\mathbf{A}$, and $U = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m]$, $V = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n]$, $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$ ($\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n$) are called the left (right) singular vectors of $\mathbf{A}$. [5]

It turns out that the SVD of the matrix $\mathbf{A}$ can provide solution to the orthogonal clustering of the row or column vectors of $\mathbf{A}$.

THEOREM The left (right) singular vectors of $\mathbf{A}$ are the cluster vectors discovered through orthogonal clustering of row (column) vectors of $\mathbf{A}$.

PROOF Since $\mathbf{A}\mathbf{A}^T$ is a $m \times m$ symmetrical matrix, we can compute Rayleigh Quotient of $\mathbf{A}\mathbf{A}^T$ w.r.t $\mathbf{x} \in R^m$:

$$R(\mathbf{x}) = \frac{\mathbf{x}^T (\mathbf{A}\mathbf{A}^T)\mathbf{x}}{\mathbf{x}^T\mathbf{x}} = \frac{(\mathbf{x}^T\mathbf{A})(\mathbf{A}^T\mathbf{x})}{\mathbf{x}^T\mathbf{x}} = \frac{(\mathbf{x}^T\mathbf{A})(\mathbf{x}^T\mathbf{A})^T}{\mathbf{x}^T\mathbf{x}}$$

$$= \frac{(\mathbf{x}^T\mathbf{A}, \mathbf{x}^T\mathbf{A})}{(\mathbf{x}, \mathbf{x})} = \frac{|\mathbf{x}^T\mathbf{A}|^2}{|\mathbf{x}|^2} .$$

If $\mathbf{c}_q$ represent a cluster of row vectors of $\mathbf{A}$, then $|\mathbf{c}_g| = 1$,

$R(\mathbf{c}_g) = \frac{|\mathbf{c}_g{}^T\mathbf{A}|^2}{|\mathbf{c}_g|^2} = |\mathbf{c}_g{}^T\mathbf{A}|^2$. So $|\mathbf{c}_g{}^T\mathbf{A}| = \sqrt{R(\mathbf{c}_g)}$, i.e., the cluster density of $\mathbf{c}_q$ is actually the square root of the Rayleigh Quotient of of $\mathbf{A}\mathbf{A}^T$ w.r.t $\mathbf{c}_q$. According to the definition of orthogonal clustering, $\mathbf{c}_q$ should has maximum density subject to being orthogonal to $\mathbf{c}_1, \cdots, \mathbf{c}_{q-1}$. From the above theorem about Rayleigh Quotient, and noting $L^{\perp}(\mathbf{p}_1, \cdots, \mathbf{p}_g) = L(\mathbf{p}_{g+1}, \cdots, \mathbf{p}_m)$, it is clear that $\mathbf{c}_q$ must be the g-th eigenvector $\mathbf{p}_q$ of $\mathbf{A}\mathbf{A}^T$, or the g-th left singular vector $\mathbf{x}_q$ of $\mathbf{A}$. The proof for the clustering of $\mathbf{A}$'s column vectors is similar.

Since there may be some negative elements in the cluster vectors, we add a constraint for each cluster $\mathbf{x}_q$ that $\sum_{i=1}^{m} \mathbf{x}_g(i) \geq 0$, or we use $-\mathbf{x}_q$ instead.

Then we address the problem of how to determine the appropriate cluster numbers.

DEFINITION In the previous problem setting, the cluster matrix of $\mathbf{x}_q$ is $\mathbf{X}_g = \mathbf{x}_g (\mathbf{x}_g{}^T\mathbf{A})$, similarly the cluster matrix of $\mathbf{y}_q$ is $\mathbf{Y}_q = (\mathbf{A}\mathbf{y}_q)\mathbf{y}_q{}^T$.

The cluster matrix actually represents its corresponding part in the original data matrix $\mathbf{A}$.

THEOREM $\mathbf{A} = \sum_{k=1}^{r} (s_k \mathbf{x}_k \mathbf{y}_k^T)$.

THEOREM $\mathbf{X}_q = \mathbf{Y}_q = \mathbf{C}_q = s_q \mathbf{x}_q \mathbf{y}_q^T$.

PROOF $\mathbf{X}_g = \mathbf{x}_g (\mathbf{x}_g{}^T\mathbf{A}) = \mathbf{x}_g \left(\mathbf{x}_g{}^T \left(\sum_{k=1}^{r} (s_k \mathbf{x}_k \mathbf{y}_k^T) \right) \right)$

$= \mathbf{x}_g \left(\sum_{k=1}^{r} (s_k \mathbf{x}_g{}^T \mathbf{x}_k \mathbf{y}_k^T) \right)$. Because $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$ are orthonormal eigenvectors, $\mathbf{x}_g{}^T\mathbf{x}_k = \begin{cases} 0, \text{ if } g \neq k \\ 1, \text{ if } g = k \end{cases}$,

$\sum_{k=1}^{r} (\mathbf{x}_q{}^T \mathbf{x}_k s_k \mathbf{y}_k{}^T) = \mathbf{x}_q{}^T \mathbf{x}_q s_q \mathbf{y}_q{}^T = s_q \mathbf{y}_q{}^T$, so

$\mathbf{X}_q = \mathbf{x}_q s_q \mathbf{y}_q^T = s_q \mathbf{x}_q \mathbf{y}_q^T$.

Similarly $\mathbf{Y}_q = (\mathbf{A}\mathbf{y}_q)\mathbf{y}_q^T = s_q \mathbf{x}_q \mathbf{y}_q^T$.

From this theorem, we can find out that the clusters described respectively by $\mathbf{x}_q$ and $\mathbf{y}_q$ in fact are describing the same topic.

Let $\mathbf{A}_k = \sum_{g=1}^{k} \mathbf{C}_g = \sum_{g=1}^{k} \left(s_g \mathbf{x}_g \mathbf{y}_g^T \right)$, the quality of orthogonal clustering $\mathbf{x}_1,...,\mathbf{x}_k$ ($\mathbf{y}_1,...,\mathbf{y}_k$) can be reflected by the ratio of $\mathbf{A}_k$ over $\mathbf{A}$.

DEFINITION In the previous problem setting, the *k*-step orthogonal clustering quality of $\mathbf{A}$ is

$$q(\mathbf{A},k) = \frac{\|\mathbf{A}_k\|_F}{\|\mathbf{A}\|_F} = \frac{\sqrt{\sum_{g=1}^{k} \left(s_g^2 \right)}}{\sqrt{\sum_{g=1}^{r} \left(s_g^2 \right)}}, \quad (1 \le k \le r).$$

Given a cluster-quality threshold q^* (e.g. 80%), the ideal cluster number k^* is the minimum number k satisfying $q(\mathbf{A},k) \ge q^*$.

SHOC applies orthogonal clustering to the term-document matrix of Web search results. Given a cluster-strength threshold t, the *g*-th document cluster V_q is composed of the documents whose value in vector $\mathbf{y}_q$ greater than t. The term (key phrase) with largest value in $\mathbf{x}_q$ can be taken as the label of V_q.

For example, Figure 11 contains the titles of some papers from ACM SIGIR2001 conference (http://www.sigir2001.org/), Figure 12 shows the generated term-document matrix $\mathbf{A}$. Assuming the cluster-quality threshold $q^* = 80\%$, the appropriate cluster number is 2, because $q(\mathbf{A},2) = \frac{\|\mathbf{A}_2\|_F}{\|\mathbf{A}\|_F} = \frac{\sqrt{3.21^2 + 2.61^2}}{\sqrt{26}}$ $= 0.81$,. The SVD of $\mathbf{A}$ gives the following results:

$\mathbf{x}_1 = (0.76, 0.33, 0.32, 0.26, 0.33, 0.11, 0.15)^T$,

$\mathbf{x}_2 = (-0.25, 0.17, 0.04, -0.09, -0.10, 0.61, 0.72)^T$;

$\mathbf{y}_1 = (0.34, 0.34, 0.52, 0.34, 0.08, 0.18, 0.32, 0.24,$
$0.20, 0.08, 0.15, 0.34)^T$,

$\mathbf{y}_2 = (-0.03, -0.08, -0.10, -0.13, 0.51, 0.57, -0.13,$
$-0.10, -0.02, 0.51, 0.29, -0.08)^T$.

Assuming the cluster-strength threshold $t = 0.15$, then the two document clusters are as follows.

$\mathbf{V}_1 = \{$ D1, D2, D3, D4, D6, D7, D8, D9, D11, D12 $\}$ with the term "*Summarizatio*" as its label.

$\mathbf{V}_2 = \{$ D5, D6, D10, D11, D12 $\}$,

with the term "*Language Model*" as its label.

Paper Session 1A: Summarization 1

D1: Applying *Summarization* Techniques for Term Selection in *Relevance* Feedback

D2: Temporal *Summaries* of News *Topics*

D3: *Generic Text Summarization* Using *Relevance* Measure and Latent Semantic Analysis

D4: A New Approach to Unsupervised *Text Summarization*

Paper Session 3 : Language Models 1

D5: Document *Language Models*, Query *Models*, and Risk Minimization for Information Retrieval

D6: *Relevance*-based *Language Models*

Paper Session 5A: Summarization 2

D7: *Generic Summaries* for Indexing in Information Retrieval

D8: Automatic Generation of Concise *Summaries* of Spoken Dialogues in Unrestricted Domains

D9: Enhanced *Topic* Distillation using *Text*, Markup Tags, and Hyperlinks

Paper Session 8A : Language Models 2

D10: A Study of Smoothing Methods for *Language Models* Applied to ad hoc Information Retrieval

D11: *Topic* Segmentation with an Aspect Hidden Markov *Model*

D12: Finding *Topic* Words for Hierarchical *Summarization*

Figure 11, the titles of some papers from the ACM SIGIR2001 conference.

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
Summarization	1	1	1	1	0	0	1	1	0	0	0	1
Relevance	1	0	1	0	0	1	0	0	0	0	0	0
Topic	0	1	0	0	0	0	0	0	1	0	1	1
Generic	0	0	1	0	0	0	1	0	0	0	0	0
Text	0	0	1	1	0	0	0	0	1	0	0	0
Language	0	0	0	0	1	1	0	0	0	1	0	0
Model	0	0	0	0	1	1	0	0	0	1	1	0

Figure 12, the term-document matrix of the document collections in Fighre 11.

There are some efficient algorithms which can do SVD for large sparse matrix very quickly [15]. To save time further, we can run SVD on the top-*n* items in the search results returned by search engines, then " fold-in" the rest documents incrementally [1]. Because search engines usually place high quality documents at the top of the result list, this approximation would not seriously hurt the clustering quality.

P. Drineas et al. introduced the initial " orthogonal clustering" concept [15]. This paper gives strict mathematical proof on why SVD provides solution to orthogonal clustering. J. Kleinberg pointed out that SVD on the hyperlink structure matrix can be used for ranking and clustering Web pages [16].

Furthermore, SHOC organizes the clusters of Web search results into a tree structure to facilitate browsing. This is done by checking each pair of clusters, X and Y, to see if then can be merged into one cluster or be treated as a parent-child relationship, as shown in Figure 13.

```
if ( |X∩Y| / |X∪Y| > t1 ) {
        X and Y are merged into one cluster;
}
else {
        if ( |X| > |Y| ) {
                if ( |X∩Y| / |Y| > t2 ) {
                        let Y become X's child;
                }
        }
        else {
                if ( |X∩Y| / |X| > t2 ) {
                        let X become Y's child;
                }
        }
}
```

Figure 13, the procedure to combine two clusters.

When two base clusters, X and Y are merged into one cluster, their phrase labels *label_x* and *label_y* should also be merged into one phrase *label_xy*, as the following procedure in Figure 14.

```
if ( label_x is a substring of label_y ) {
        label_xy = label_y;
}
else if ( label_y is a substring of label_x ) {
        label_xy = label_x;
}
else {
        label_xy = " label_x + label_y ";
}
```

Figure 14, the procedure to merge the labels of two clusters.

We iteratively check every pair of base clusters and organize them. A hierarchy of cluster appears at last.

One superiority of SHOC is that users are able to adjust the meaningful clustering parameters to fulfill their own needs.

4. PROTOTYPE SYSTEM

Based on the SHOC approach, we have created a prototype system named WICE (Web Information Clustering Engine). It can automatically organize multilingual Web search results in a semantic, hierarchical, and online way. Currently it only clusters search results from Google (http://www.google.com/).

WICE has demonstrated the effectiveness of our SHOC approach. Figure 15 shows the output of WICE for the query "**面向对象**" (object oriented), including clusters labeled "**面向对象程序设计**"(object oriented programming), "**面向对象分析**"(object oriented analysis), etc.

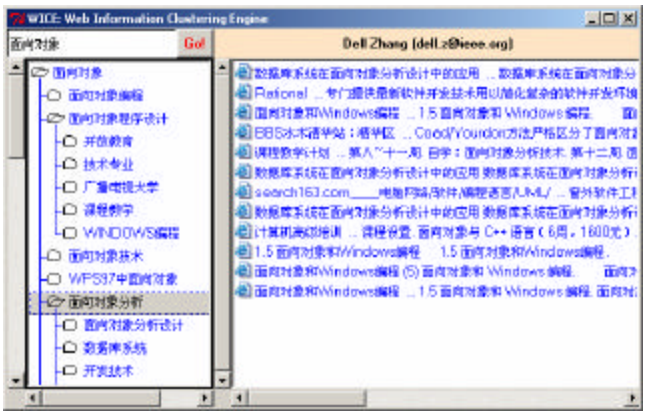

Figure 15, the output of WICE for the query "面向对象" (object oriented).

5. CONCLUSION

The main contributions of this paper include the following. (1) The benefits of using key phrases as natural language information features are discussed. An effective and efficient algorithm based on suffix array for key phrase discovery is presented. The efficiency of this method is very high no matter how large the language's alphabet is. (2) The concept of orthogonal clustering is proposed for general clustering problems. The reason why matrix SVD (Singular Value Decomposition) can provide solution to orthogonal clustering is strictly proved. The orthogonal clustering algorithm has a solid mathematics foundation and many advantages over traditional heuristic clustering algorithms. (3) The WICE system is designed and implemented to automatically organize multilingual Web search results through a semantic, hierarchical, online clustering approach named SHOC.

Our prototype system suggests that clustering of Web search results can really help users find relevant Web information more easily and quickly. Extensive experiments are needed to evaluate the performance of the proposed SHOC approach.

REFERENCES

[1] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. SIAM Review 37 (4), 573-595, 1995.

[2] L. F. Chien, PAT-Tree-Based Keyword Extraction for Chinese Information Retrieval, In Proceedings of the 20th ACM SIGIR International Conference on Information Retrieval, 1997.

[3] D. Cutting, D. Karger, J. Pedersen, J. W. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large

Document Collections. In Proceedings of the 15th Annual International ACM/SIGIR Conference, Copenhagen, 1992.

[4] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, Indexing by latent semantic analysis, Journal of the American Society for Information Science, 41, pp. 391-407, 1990.

[5] G. Golub and C. V. Loan, Matrix Computations, second edition, Johns-Hopkins, Baltimore, 1989.

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

[7] C. Kwok, O. Etzioni, and D. S. Weld. Scaling Question Answering to the Web. In Proceedings of the Tenth International World Wide Web Conference, Hong Kong , May 2001.

[8] R. Kannan, S. Vempala and A. Vetta. On clusterings: good, bad and spectral. In Proc. 41st Symposium on the Foundations of Computer Science, 2000.

[9] U. Manber and E. Myers. Suffix arrays: A new method for on-line string searches. In Proceedings of the first Annual ACM-SIAM Symposium on Discrete Algorithms, pages 319-327, 1990.

[10] Y. Wang, M. Kitsuregawa. Link-based Clustering of Web Search Results. In Proceedings of The Second International Conference on Web-Age Information Management (WAIM2001), Xi'An, P.R.China, Springer-Verlag LNCS, July, 2001.

[11] M. Yamamoto and K. W. Church. Using Suffix Arrays to compute Term Frequency and Document Frequency for All Substrings in a Corpus. Computational Linguistics, vol 27:1, pp.1-30, MIT Press, 2001.

[12] O. Zamir and O. Etzioni. Web Document Clustering: A Feasibility Demonstration. In Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval, Melbourne, Australia, 1998.

[13] L. F. Chien. PAT-tree-based adaptive keyphrase extraction for intelligent Chinese information retrieval. Information Processing and Management, 35(4), pp.501-521, 1999.

[14] C. H. Chang. and S. C. Lui. IEPAD: Information Extraction based on Pattern Discovery. In Proceedings of the tenth International Conference on World Wide Web, Hong Kong, May 2-6, 2001.

[15] P. Drineas, A. Frieze, R. Kannan, S. Vempala and V. Vinay. Clustering in large graphs and matrices. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[16] J. Kleinberg. Authoritative sources in a hyperlinked environment. Journal of the ACM, (46), 1999.