6-20-2004

Identifying variable-length meaningful phrases with correlation functions

Hyoung-Rae Kim

Philip K. Chan

Identifying Variable-Length Meaningful Phrases with Correlation Functions

Hyoung-rae Kim and Philip K. Chan
Department of Computer Sciences Technical Report CS-2004-10,
Florida Institute of Technology
Melbourne, FL 32901, USA
hokim@fit.edu, pkc@cs.fit.edu

ABSTRACT
Finding meaningful phrases in a document has been studied in various information retrieval systems in order to improve the performance. Many previous statistical phrase finding methods had different aim such as document classification. Some are hybridized with statistical and syntactic grammatical methods; others use correlation heuristics between words. We propose a new phrase-finding algorithm that adds correlated words one by one to the phrases found in the previous stage, maintaining high correlation within a phrase. Our results indicate that our algorithm finds more meaningful phrases than an existing algorithm. Furthermore, the previous algorithm could be improved by applying different correlation functions.

Categories and Subject Descriptors
H.3.3 [**Information Storage and Retrieval**]:Information Search and Retrieval –*information filtering, selection process.*

General Terms
Algorithms.

Keywords
phrase, variable-length phrase finding algorithm.

1. INTRODUCTION
Statistical phrase finding algorithms are mainly used for improving the performance of information retrieval [5,6,19]. There are three main approaches: syntactic [12,14], statistical [14], and hybridized [8]. Our research mainly focuses on the statistical approach, which does not need any grammatical knowledge and has easy adaptability to other languages. Statistical phrase-finding approaches have been used for expanding vector dimensions in clustering multiple documents [19,20], or finding more descriptive or important/meaningful phrases [1,2]. This paper compares previous statistical approaches and attempts to find meaningful phrases in a document.

Ahonen et al [1], Zamir and Etzioni [21], and Chan [2] introduced phrase-finding algorithms. Ahonen's algorithm depends on conditional probability and needs a fixed maximum phrase length. We suspect that the use of two parameters – a threshold to remove less descriptive phrases in generating stage and maximum phrase length – are too strict. Zamir and Etzioni [21] introduced a fast algorithm, but it uses only frequency information. Chan's algorithm [2] improved the performance by using correlation information within a phrase. However, Chan's algorithm can generate non-existing phrases and is vulnerable to synthetic data.

The definition of meaningful is unique to each individual. So, we define a phrase as more meaningful if it is meaningful to the most people. We let each individual define his or her own definition of meaningful. We propose two variable-length phrase-finding algorithms, VPF-1 and VPF-2, which find more meaningful phrases. VPF-1 is designed to remove the maximum length of phrases in Ahonen's algorithm, and VPF-2 is designed to fix the problem in Chan's algorithm by combining it with VPF-1. Both algorithms add correlated words one by one to the phrases made in the previous stage. Both apply pruning to remove less meaningful phrases.

The main contributions are:

(1) We proposed two variable-length phrase-finding algorithms that is designed for finding meaningful phrases;
(2) The time complexity remains as O(N) where N is the size of the input sequence under a specified condition;
(3) These algorithms do not need any user-specified parameters;
(4) The algorithms achieve improved performance by pruning less meaningful phrases;
(5) More meaningful phrases than previous methods are found and the improvement in performance is statistically significant;
(6) Some correlation functions are prominent by being ranked high with both algorithms;
(7) Ahonen's algorithm is improved by applying different correlation functions.

The rest of this paper is as follows: Section 2 presents related work regarding statistical phrase-finding methods; Section 3 discusses input and output and compares our approach with previous algorithms; Section 4 provides detailed description of our variable-length meaningful phrase-finding algorithms (VPF-1 and VPF-2); Section 5 describes the desirable properties of correlation functions and lists all correlation functions we used; Section 6 discusses about experiment; Section 7 presents and analyzes our results; Section 8 summarizes out work.

2. RELATED RESEARCH

There are two main approaches to finding phrases. The first is related to clustering documents and retrieving documents that most likely match the user's information need. This research focuses on which words and phrases are more important in clustering documents. The other attempts to find phrases meaningful to human users.

Wu and Gunopulos [20] examined the usefulness of phrases as terms in vector-based document classification. They used statistical techniques to extract phrases from documents whose document frequency (*df*) is larger than or at least equal to a predefined threshold. Fagan [6] selected phrases having a document frequency of at least 55 and a high co-occurrence in the same sentence. Mitra et al. [14] collected all pairs of non-function words that occur contiguously in at least 25 documents. Turpin and Moffat [19] used Mitra's method for statistical phrases for vector-space retrieval. Since the aim of these approaches is to find the significant words or phrases among documents, they remove common words or phrases among documents, which could also remove meaningful phrases in a document. Furthermore, only two-word phrases are considered, whereas ours has no limitation in phrase length.

Croft, et al. [5] describe an approach where phrases identified in natural language queries are used to build structured queries for a probabilistic retrieval model and showed that using phrases could improve performance. They used *tf***idf* information for a similarity measure. Croft [4] segmented a document's text using a number of phrase separators such as verbs, numbers, dates, title words, company designators, format changes, etc. Next, his method checks the candidate phrases to see if they are syntactically correct. Finally, the occurrence frequency of the remaining phrases is checked. Our paper mainly focuses on a statistical approach without introducing a syntactic method. We uses simple phrase separators (i.e., stop-words and non-alphabet characters), which generalizes our method independent from a certain language.

Gokcay and Gokcay [8] used statistically extracted keywords and phrases for title generation. Their statistical method used grammatical information of tags and sentences, but it is hard to determine a sentence without grammatical information. They used cosine correlation function for comparing the similarity of two words. Our research experimentally shows which correlation functions are better than others in terms of measuring word correlation. Ahonen et al. [1] applied Mannila and Toivonen's [13] algorithm for finding phrases. In this algorithm a user has to specify the maximum phrase length and certain thresholds. Our algorithm does not need a specified maximum phrase length.

3. PROBLEM

We desire to find phrases meaningful to human users. The goal will be to devise an algorithm that improves the number of matching phrases to these phrases selected by a human subject. The term "meaningful phrase" is the phrase that satisfies a higher % of the individual's definition of meaningful. The input and output data can be specified as:

```
Input: a sequence of words (a document)
Output: a set of meaningful phrases
```

Ahonen et al [1], Zamir and Etzioni [21], and Chan [2] introduced phrase-finding methods. Ahonen's method finds all possible combinations of words within a fixed window. Suppose the window size is 6 and the string in that window is "abcdef". Their algorithm generates all possible cases: "ab", "bc", "cd", "de", "ef", "abc", "bcd", "def", … "bcdef", "abcdef". Then, it computes the conditional probability for the weight of those phrases. A phrase "abc" has two possible weights from *P*("c"| "ab") and *P*("bc"| "a"), from which the higher value is chosen. Even with the algorithm's exhaustive examination, its performance is, as will be shown later, lower than Chan's [2].

Zamir's [21] has linear time complexity. The critical drawback of Zamir's algorithm is that their algorithm uses only frequency information. They build the suffix tree based on the overlap of words (frequency) and then collect neither too frequent nor too rare phrases. Suppose we are collecting a phrase "abcd", Zamir's method uses only the frequency of the term. This method needs two user defined parameters: one for removing too rare or too frequent words and the other for selecting phrases out of all possible phrases.

Chan's phrase-finding algorithm calculates the correlation values of all pairs. For instance, for a phrase "abcd", it calculates all correlations of pairs: *Corr*("a","b",0), *Corr*("a","c",1), *Corr*("a","d",2), *Corr*("b","c",0), *Corr*("b","d",1), and *Corr*("c","d",0). The function *Corr* gets three variables - two words and the distance between the two words. The main drawback of this method resides in its incompleteness. Suppose there is a string *S*="abaxbxaxxbxaxxxb…xxbbxbxbxxbxxxb…", where 'a' and 'b' represent words, and 'x' represents any word. If all correlations between 'a' and 'b' with 1 through 4 distances, and correlations between 'b' and 'b' with 1 through 4 distances have value of higher than the threshold, Chan's algorithm will generate a word "abbbb" that does not exist in the string. This problem also makes the quality of phrases suspicious, because correlation values for each pair of words within a phrase can be affected by other phrases. For example, the correlation value of the non-existing word "abbbb" could be high, if the correlation values for all pairs are high. Moreover, if the correlations between 'a' and 'a' with 1 through 4 distances (like "..xxaaxaxxaxxxa..") are higher than the threshold, then the algorithm will generate 25 possible combinations of 'a' and 'b' with length of 5 (e.g., "aaaaa", "aaaab", "aaaba", … "bbbba", "bbbbb"). These cases are very unlikely to happen in a normal article such as newspaper or journal article. But web pages contain lists of similar product names or tables that just arrange a few different repeating words many times. We experienced these non-existing words in our experiment such as "test pass test", "student test pass", "teach assist teach class", etc. Another disadvantage of Chan's algorithm is that it requires a user-defined maximum phrase length. Our method is similar to Chan's original method. We added the correlations *Corr*("ab","c",0) and *Corr*("abc","d",0) in addition to Chan's in the case of the running example above.

One might insist that each phrase $P\{m\}$ of length m has the form $P\{m\text{-}1\}w$, where w is one word and $P\{m\text{-}1\}$ is a phrase of length m-1. Since the phrase $P\{m\text{-}1\}w$ is defined by the correlation between $P\{m\text{-}1\}$ and w, it is possible that the correlation exists between a non-phrase $P\{m\text{-}1\}$ and a word w.

That is, $P\{m\text{-}1\}$ is not a phrase, but can be extended into a phrase of length m. If this is possible, it is also possible that even if there exists a phrase, $P\{m\}$, in a document, the phrase $P\{m\}$, could not be generated because $P\{m\text{-}1\}$ does not exist. For example, there exists a phrase "wireless powerful computer" in a web page. But, since "wireless powerful" is not a phrase, it is possible that the phrase could not be generated. However, if the phrase is meaningful/important enough, the sub phrases "wireless powerful" and "powerful computer" will be generated. Next, "computer" will be added to "wireless powerful". To relieve this problem, we calculate the threshold once at the beginning – this means the threshold is consistent. If the correlation value of "wireless powerful" is lower than the value of "wireless powerful computer" then the shorter phrase will be removed at a pruning stage. Ahone [1] generates all possible phrases within a specified length, which prevents this problem from happening. In contrast, our method emphasizes inner correlation of a phrase - if a phrase has low inner correlation our method does not consider that phrase meaningful.

4. APPROACH

Our algorithm consists of two components: the main algorithm and the correlation function. In this section we describe the main algorithm. In preparation for our phrase-finding algorithm, we extract words from a web page visited by the user, filter them through a stop list, and stem them [1,2,7,21]. Our original implementation suffers from the exponential time complexity with large numbers of different words. We explain different ways of implementation, and compare both implementation strategies.

4.1 VPFs Algorithms

Our algorithm receives a sequence of words as input and returns meaningful phrases. It combines words into phrases until it generates no more phrases. Figure 1 illustrates the pseudo code for the Variable-length phrase-finding algorithm (VPF). VPF uses three functions - *FindPhrase*, *Corr*, and *Check* - and three main variables - *List*, *thre*, and *Hash*. The *List* variable stores all collected phrases (in *PList* attribute). All correlation values between selected pair of words are stored in *Hash* table. We originally stored all correlation values generated with the first word in a phrase in *PList* and all distinct words in *PList*. The problem resided in its computation of the correlation of unnecessary pairs. This algorithm uses only the phrases that exist in the example input sequence. An element of the *PList* attribute consists of multiple words ($w_1..w_n$), has its position list in *posi* attribute, and keeps its inner correlation value in *sim* attribute. The *thre* variable keeps the calculated threshold value that differentiates "strong" relations from "weak" relations.

All 1-gram distinct words are stored in *List*[1].*PList* and 2-gram distinct words are in *List*[2].*PList* while we are scanning a sequence. Then the CalculateThreshold function calculates threshold using *List*[2].*PList*:

```
thre←Average of {Corr(w1.posi,w2.posi,0) |
      w1+w2 ∈ List[2].PList}
```

The *List* and *thre* data are passed into the *FindPhrase* procedure which collects all variable-length phrases. Once the phrases are acquired, they are pruned. The *PrunPhrase* function simply removes all sub-phrases which have a *sim* value lower than the *sim* value of the super-phrases.

The *FindPhrase* procedure uses two variables: a temporary variable *Nseq* that keeps a new sequence and *N* that is the length of phrases (initial value is 2). The *Hash* variable is initialized as *NULL*. *FindPhrase* then creates new sequence using the previous *PList*.

The *CreateNewSequence* function lists all distinct words in *PList* and sorts them by positions. Because we already collected all 2-grams, if *N* is 2 then it does nothing but returns *Null*, and the *CollectNGram* function returns *List*[2].*PList*. The *CollectNGram* function gets *Nseq* as input, scans it for *n*-grams, and stores them in *PList*. The input sequence can have distance or gaps between positions, because the sequence is made by only the words in the previous *PList*. Then it collects only valid pairs having no gap in a *n*-gram. The *for* loop checks all phrases in *PList* and removes phrases with low correlation values (*sim*) from *PList* using *Check* function. The *FindPhrase* procedure recursively increases the phrase length until no new phrases are generated. The *Nseq* variable is a temporary variable, where all distinct words that exist in the phrases collected in the previous stage are sorted by their position. The time complexity of building *Nseq* will be $O(S \log S)$, where *S* is the size of a sequence. The process of building this *Nseq* is the most expensive. However, when we sort distinct words in *List*[2].*PList*, many of the words are already removed. Therefore, we can claim that the time complexity of building *List*[*j*].*PList* is:

$O(S)$, where *S* is the sequence size,

under the condition of $S_{i+1} \log S_{i+1} < S_i$. What properties of an article keep this condition will be our future work.

There are two VPFs: VPF-1 and VPF-2. The only difference between the two is that VPF-2 cross-references more correlations than VPF-1. *Check* in Figure 2 calculates the correlation value between two events and returns a true or false response depending on whether their correlation is "strong" or "weak". If we run VPF-1, then *Check* calculates the correlation between two events, $p[1..n\text{-}1]$ and $p[n]$, and keeps the correlation value in the *sim* property of the phrase. For example, if *p*= "computer science seminar", then $p[1..n\text{-}1]$= "computer science" and $p[n]$= "seminar". If we run VPF-2, then *Check* measures the inner correlation value of a phrase.

The *for* loop in Figure 2 checks to see if any of the correlation values between a pair of words in a phrase *p* are lower than the calculated threshold. When we calculate a correlation value for phrase *p*, it needs only (*n*-1) number of comparisons, where *n* is the length of a phrase *p*. For example, if we calculate the inner correlation of "abcd", then the only calculations we need are *Corr*("a","d",2), *Corr*("b","d",1), and *Corr*("c","d",0). Suppose we store the sum in *temp_sum*. In this case, the time complexity is O(*n*), where *n* is the length of a phrase *p*. The correlation for a phrase *p* is calculated as:

```
p.sim←(old_p.sim × Combinatorial(n-1,2)+
      temp_sum)/ Combinatorial(n,2),
```

where, *old_p* is $p[1..n\text{-}1]$ and *temp_sum* is the sum of newly calculated correlation values. The *item* method in *Hash* searches for the corresponding key and return the correlation value. The *add* method adds a key and a correlation value to the *Hash* table. The *Key* was composed of two words and distance.

```
Input: Example- a sequence of words
Output: Collected phrases
Procedure VPF (Example)
 List- array of lists that store all phrases
 thre- threshold value
 Hash- stores correlation value
1.  List[1].PList← 1-gram with position
2.  List[2].PList← 2-gram with position
3.  thre←CalculateThreshold(List[2].PList)
4.  FindPhrase(List, Null, 2, thre)
5.  PrunPhrase(List)
6.  return all phrases in List
End Procedure

Input: List- store all phrases
 Hash- store correlation value
 N- length of phrase, initial value is 2
 thre- threshold value
FindPhrase(List, Hash, N, thre)
1.  NSeq←CreateNewSequence(List[N-1].PList)
2.  List[N].PList←CollectNGram(Nseq)
3.  for each p in List[N].PList
4.    if Check(Hash,p,N,thre)=True then
5.      recount positions of p
6.    else
7.      remove p from PhraseList
8.    end if
9.  next
10. FindPhrase(List, Hash, N+1, thre)
End Procedure
```

Figure 1. VPFs algorithm

```
Input: Hash- store correlation value
 p- a phrase
 n- length of a phrase
 thre- threshold value
Function Check (Hash, p, n, thre)
1.  // if we run VPF-1 then
2.  p.sim←Corr(p[1..n-1],p[n],n-2)
3.  if p.sim > thre then
4.    return true
5.  end if
6.  // if we run VPF-2 then check this also
7.  p.sim←Calculate new correlation value
8.  for each wj in p step j=[1..n-1]
9.    temp←Hash.item(Key(wj,p[n],n-j-1))
10.   if temp < thre or temp=NULL then
11.     return false // check inner corr
    value
12.   end if
13. next
14. Hash.add(Key(p[1],p[n],n-2),
          Corr(p[1],p[n],n-2)
15. return true
End Function
```

Figure 2. Check function

```
Input: T1, T2- array of position occurred
 Dist- distance
Output: correlation value between T1 and T2
Function Corr (T1,T2,Dist)
1.  A←CalcuPrePercentage(T1)
2.  B←CalcuPostPercentage(T2)
3.  A∩B←Intersection(T1,T2,Dist)/(page size-
    1)
4.  return CorrelationFunction(A,B,A∩B)
End Function
```

Figure 3. Corr function (calculate correlation)

Corr in Figure 3 calculates a correlation value of events. The inputs are two arrays of positions (T_1 and T_2) and distance/gap (*dist*) – note that the input only receives the position information from the input event. It calculates pre- and post-percentages of each events. The way the two percentages are calculated is detailed in the next section. We can also apply various correlation functions in the place of *CorrelationFunction*. We apply 32 different correlation functions. The *Intersection* counts all adjacent points based on the distance in time O(T), where T is the maximum distance between T_1 and T_2.

$$Intersection = Count\ \{a+1 = b - \text{dist} \mid a \in T_1 \text{ and } b \in T_2\}$$

The time complexity of this implementation of VPFs consists of four operations: the number of recursion, which is equivalent to the maximum phrase length generated (L); the time required building *Nseq* (S). The *Corr* function has O(T) time complexity, where T is the position length. *Check* has O(L) time complexity, where L is the phrase length. The time complexity of VPFs in general case is roughly:

O(S), where S is the sequence size.

4.2 Calculating Pre- and Post-Percentage

This section explains mainly the *CalculatePrePercentage* and *CalculatePostPercentage* in *Corr* function in Figure 3. Correlation function uses three probability values: P(A), P(B), P(A,B). In order to improve the phrase-selection accuracy, we need to calculate for each word the percentage that a word can come before any other words and the percentage that the word can come after any other words, called pre-percentage and post-percentage respectively. The idea is that if a word occurred at the end of a sequence, then this word lose one chance to come before any other words, so we adjust the pre-percentage of the word by deducting one from the # of occurrence of a word. The post-percentage is vice versa. This adjustment is minor in a lengthy document, but it is significant in a shorter document.

Suppose we have a sequence of five words:

"banana ice-cream banana ice-cream spoon".

A simple way to calculate the probability of each word and pair: $P(w_1)$=2/5, $P(w_2)$=2/5, and $P(w_3)$=1/5, where w_1= "banana", w_2= "ice-cream", w_3= "spoon", and the total page size is 5. The intersections are $P(w_1,w_2)$=2/4, $P(w_2,w_3)$=1/4, $P(w_1,w_3)$=0, etc., where the total possible number of phrases is 4. However, when we calculate $P(w_1,w_2')$, it produces an erroneous result by yielding a negative value ($P(w_1)$-$P(w_1,w_2)$ = –1/10 = 2/5-2/4). This example shows the need for calculating pre- and post-percentages.

We can view a string S[1..n] of n consecutive words as two sub strings, S_{pre}=S[1..n-1] and S_{post}=S[2..n]. Pre- and post-percentage of w can be computed in time O(1), when we know all the positions where w occurred:

$$w_{\text{pre-percentage}} = \text{Frequency of } w \text{ in } S_{pre} \,/\, |S_{pre}|$$
$$w_{\text{post-percentage}} = \text{Frequency of } w \text{ in } S_{post} \,/\, |S_{post}|.$$

With the previous example, the pre-percentage of w_1 "banana" is (2/4) where w_1 occurred two times; the post-percentage is (1/4) where w_1 occurred one time in S_{post}. For example,

$P(w_1,w_2')$ becomes 0 (=1/4–1/4) and the result is very reasonable because "banana" and "ice-cream" are adjacent in the sequence.

5. CORRELATION FUNCTIONS

The VPF algorithms build phrases and correlation functions actually calculate the weight of a phrase. Correlation function is important in terms of selecting more meaningful phrases. The VPF is able to cooperate with many different existing correlation functions, and it can be hard to choose one correlation function out of many. In this section, we describe several key properties of a good correlation function.

Much of the statistical work in building multi-word features focuses on co-occurrence [3,16]. All correlation measures are not equally good at capturing the dependencies between variables. It is because each correlation function has its own bias in preferring a set of diagrams to another. Those dependencies can be described in a Venn diagram as shown in a feature of A, B, and $A \cap B$.

Piatetsky-Shapiro [15] has proposed three key properties that a good correlation function, F, should satisfy:

P1: if A and B are statistically independent, then F is 0;
P2: F monotonically increases with $P(A,B)$ when $P(A)$ and $P(B)$ remain the same;
P3: if P(A) (or P(B)) increases when the rest of the parameters (P(A,B) and P(B) (or P(A))) remain unchanged, then F monotonically decreases.

Statistical independence can be measured by the determinant operator, where $Det(A,B) = A \cap B \times A' \cap B' - A \cap B' \times A' \cap B$. Thus, a singular diagram D is independent when its determinant is equal to zero [17]. Another important operation in finding phrases is distinguishing between positive and negative correlations ($P4$). Measuring their *cross product ratio* (*CPR*) can assess the significance of the correlation between A and B [16] and is defined as:

$$\log CRP(A,B) = \log \frac{P(A,B)P(A',B')}{P(A',B)P(A,B')}$$

Negative correlation has a negative *log* CPR value. $P4$ is that F can distinguish positive and negative correlation of A and B. Tan et al. [17] described this property in detail. Since positive correlation is much more important than negative correlation in finding phrases, we only measured the change of correlation values over positive correlation.

Tan et al. [18] illustrated those properties and extended them to each of the existing measures to determine if the existing measure satisfies the properties required [10]. Some of these properties have been extensively investigated in the data mining literature [9,17,18]. We examined 32 correlation functions of properties and cooperated them with phrase-finding algorithms. A complete list of the correlation functions to be examined in this study is given in Table 11.

6. EXPERIMENTS

We use five New York Times articles and five Web pages collected from our department server. We chose five web pages to test, because contents in a web page differ from the content found in normal article. The data used in this study is accessible at http://my.fit.edu/~hokim/conference/phrase/dataset.pdf. The article size was about 2 pages, because each volunteer had to read 10 articles.

We asked ten human subjects, other than the authors, to read the articles and choose their top 10 meaningful phrases for each article or Web page. The instruction that we gave them were:

- Identify top 10 "meaningful/important" phrases for each article.
- Phrases are defined as two or more adjacent words that are meaningful, for example, "computer science," "florida institute of technology," ... The definition of meaningful is up to you.

We will measure the number of matches between the human subjects' selections and different correlation functions' selections as well as different phrase-finding algorithms.

We evaluate the meaningfulness of phrases. We believe the closer a match comes to our set of human-selected phrases, the better the phrase-finding algorithm is in terms of finding meaningful phrases. To evaluate the correlation functions for each phrase-finding algorithm, we have two evaluation criteria: the number of exact matches and the number of simple matches. The # of exact matches of a method is measured by the percentage of the matches between the human's and a method's (we have 160 methods = 5 algorithms × 32 correlation functions). We count each match with a human's and then average the 10 compared results.

The number of simple matches counts the matched phrases against the list collected by all human (i.e., the union of the words from the 10 human subjects). The list will be less then 100 because some phrases can overlap. Some words are more popular than the others. The counting of the # of simple match is less affected by popularity.

We also count average matching of human – in this case, we divided the sum by 9. There are cases for human or algorithm to select less than 10 phrases. In order to be fair in these cases, we use an additional adjustment function. For example, if there are 5 matches out of 10, the # of matching is 5×1/10. If there are 5 matches out of 9, then we assigned 5×1/9. But, if there are 5 matches out of 8, then we assigned 5×1/8.5. The denominator is not 8 so as to assess a small penalty for not finding 10. The generalized formula is:

$$\text{Adjustment function (m, f)} = \frac{m}{8 + \frac{2}{2^{10-f}}}$$

where m is the # of matched items and f is the # of selected items.

We also applied different correlation functions to Ahonen's algorithm to see if the difference of the performance depended on the correlation functions. Ahonen used two different similarity functions: conditionaly probability (Confidence, S11) for filtering phrases and mutual confidence (S32) for ordering the collected phrases determining which phrase is more important than the other. Since he used fixed user-defined threshold (0.2) for filtering the phrases, we only varied the correlation function used for ordering phrases.

Table 1. Comparing by top 10 best methods - exact match

with prune			without prune		
Rank	**Method**	**Avg.**	**Rank**	**Method**	**Avg.**
1	vpf-1 S25	**0.933**	**1**	vpf-1 S25	0.883
2	vpf-1 S16	**0.920**	**2**	vpf-1 S28	0.871
3	vpf-1 S28	**0.912**	**3**	vpf-1 S16	0.866
4	vpf-2 S16	**0.860**	**4**	vpf-2 S16	0.858
5	vpf-2 S29	0.851	**5**	vpf-2 S25	**0.856**
6	vpf-2 S25	0.837	**6**	vpf-2 S29	**0.850**
7	vpf-2 S28	0.832	**7**	vpf-2 S28	**0.848**
8	vpf-1 S8	0.824	**8**	vpf-1 S10	**0.825**
9	vpf-1 S10	**0.819**	**9**	vpf-1 S29	0.810
10	vpf-1 S29	**0.814**	**10**	vpf-1 S27	0.796

Table 2. Comparing by maximum cases - exact match

Method	**Prune**	**Avg.**	**a1**	**a2**	**a3**	**a4**	**a5**	**a6**	**a7**	**a8**	**a9**	**a10**
VPF-1	**Yes**	1.16	1.85	1.20	.50	.48	.89	2.00	1.10	1.18	1.00	1.40
	No	1.12	1.67	1.10	.50	.48	.89	2.00	1.10	1.18	1.00	1.30
VPF-2	**Yes**	1.23	2.00	1.20	.50	.48	.90	2.44	1.22	1.30	.80	1.40
	No	1.17	1.67	1.10	.50	.48	.90	2.00	1.22	1.30	1.20	1.30

Table 3. Ranked by average - exact match

Rank	**Method**	**Avg.**	**a1**	**a2**	**a3**	**a4**	**a5**	**a6**	**a7**	**a8**	**a9**	**a10**
1	vpf-1 S25	0.933	1.8	.8	.4	.3	.6	1.9	.2	1.0	1.0	1.3
2	vpf-1 S16	0.920	1.8	.5	.3	.5	.9	1.7	.1	1.0	1.0	1.3
3	vpf-1 S28	0.912	1.8	.8	.4	.3	.6	1.9	.2	.8	1.0	1.3
4	vpf-2 S16	0.860	1.8	.5	.3	.5	.9	1.7	.1	1.0	.4	1.3
5	chans S16	0.858	1.7	.5	.3	.5	.9	1.7	.1	.8	.8	1.3
6	chans S25	0.856	1.7	.8	.4	.3	.6	1.8	.1	.8	.8	1.3
7	vpf-2 S29	0.851	1.2	1.1	.4	.1	.7	2.0	.1	.7	.8	1.4
8	chans S29	0.850	1.1	1.0	.3	.2	.7	2.0	.1	.7	1.2	1.3
9	chans S28	0.848	1.7	.7	.4	.3	.6	1.8	.1	.8	.8	1.3
10	vpf-2 S25	0.837	1.8	.8	.4	.3	.6	1.5	.1	1.0	.5	1.3
29	ahonen S32	0.767	1.3	.8	.3	.1	.5	1.5	.2	1.0	.8	1.2
136	ahonen_gap S32	0.452	1.0	.7	.3	.1	.4	1.4	.0	.0	.7	.0

Table 4. Ahonen with other correlation function-exact match

Rank	**Method**	**Avg.**	**a1**	**a2**	**a3**	**a4**	**a5**	**a6**	**a7**	**a8**	**a9**	**a10**
18	ahonen S6	0.797	1.4	.8	.4	.1	.5	1.5	.3	.8	.9	1.2
21	ahonen S10	0.779	1.2	.8	.3	.2	.7	1.5	.3	.8	.9	1.2
21	ahonen S11	0.779	1.2	.8	.3	.2	.7	1.5	.3	.8	.9	1.2
21	ahonen S12	0.779	1.2	.8	.3	.2	.7	1.5	.3	.8	.9	1.2
21	ahonen S17	0.779	1.2	.8	.3	.2	.7	1.5	.3	.8	.9	1.2
21	ahonen S26	0.779	1.2	.8	.3	.2	.7	1.5	.3	.8	.9	1.2
26	ahonen S20	0.774	1.3	.8	.3	.1	.5	1.5	.3	.8	.9	1.2
26	ahonen S23	0.774	1.3	.8	.3	.1	.5	1.5	.3	.8	.9	1.2

7. ANALYSIS

7.1 With-Prune vs. Without-Prune

Our algorithm has a pruning function. The results differed whether we added the pruning function or not. We compared them by comparing the top 10 best methods and by comparing maximum cases.

We measure the average # of matches for each method. The method's result is compared with each human subject's selection for one article. If a phrase is selected by several subjects, every match is counted. Therefore, finding more popular phrases increases the matching average. This counting is called *exact match*.

By composing 2 algorithms and 32 correlation functions, we generated 64 methods. All methods were ordered and ranked in the order of average # of matches. We presented the result in Table 1 for with and without prune. Both cases include the top 10 methods and their average # of matches. With-prune won 6 times and without-prune won 4 times. However, the higher-ranking methods were more important than lower-ranking methods.

We selected the best combination of an algorithm and a correlation function for each article. This told us the maximum performance of an algorithm when they are combined with the best correlation function. The comparison by maximum cases was shown in Table 2. We had two algorithms: VPF-1 and VPF-2. Each algorithm had two cases with prune (Yes) and without prune (No). Then we calculated the average of all articles. Both VPFs had higher average of maximum case when they have pruning. From the observation of the two tables above, we concluded that VPFs had higher performance when they had pruning.

7.2 Analysis of Exact Match

From the above experiment we decided to use pruning in our algorithms. The following analysis is conducted over the algorithms with pruning. We first used exact match counting as before. The main purpose of the analysis in this section is to choose the best method.

7.2.1 Top 10 best methods

Which method is the best is the most interesting question. We averaged the results from 10 articles and sorted by the average to rank all 160 methods. We presented the results in Table 3 and included the rank, methods used, and the average of 10 articles. In the next 10 columns, we showed the results' average from 10 human subjects. Each method was composed of an algorithm and a correlation function. Notice that, at the bottom of the table we also presented the results of previous methods. Ahonen used correlation function S32. He also introduced a method with gaps. The row Ahonen_gap represented the results using Ahonen's method allowing gaps within a phrase.

The best method was the combination of VPF-1 and correlation functions S25 followed by S16 and S28 – all those three correlation functions satisfied Piatetsky-Shapiro's three desirable properties and distinguish positive from negative correlations. The best method VPF-1 with S25 matched 0.93 phrases on average with the phrases selected by a human subject. Interestingly, VPF-1 won the top 3. Chan's algorithm and VPF-2 occupied the next ranks. Another observation was that the correlation functions S25, S16, and S28 that marked high rank with VPF-1 also marked high rank with Chan's and VPF-2. This observation implied that the performance also depends on the correlation functions.

Table 5. Comparing with human – exact match

Method	Avg.	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
Human best	1.48	2.70	1.50	.33	1.10	.40	1.90	1.89	2.94	.50	1.50
Human worst	1.03	1.20	.44	.78	.40	.70	1.61	.70	1.10	1.22	2.18
Human avg.	1.30	1.97	.88	.60	.64	.57	1.96	1.44	2.00	1.26	1.70

Table 6. Maximum cases – exact match

Method	Avg.	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
VPF-1	1.16	1.85	1.20	.50	.48	.89	2.00	1.10	1.18	1.00	1.40
VPF-2	1.23	2.00	1.20	.50	.48	.90	2.44	1.22	1.30	.80	1.40
ahonen	.90	1.40	.85	.47	.18	.70	1.45	.50	1.00	1.00	1.50

Unfortunately, Ahonen's algorithm ranked 30 and Ahonen_gap 139. These methods matched 0.76 and 0.45 numbers of phrases with human subjects respectively. The low performance with gap is the same phenomenon as shown in the Ahonen's experiment [1]. We conducted t-Test (paired two sample for means) between VPF-1 with S25 and Ahonen with S32. There was a clear statistically significant difference between the two methods with 95% confidence (P=0.016). Therefore, we can conclude that VPF-1 with S25 found statistically significantly more meaningful phrases than Ahonen's previous algorithm.

Ahonen's algorithm with other correlation functions received higher ranks such as S6, S10, S11, S12, S17, S26, and S20 as shown in Table 4. They all ranked above 18, 21, and 26, which are higher than Ahonen's original method (29). This indicates Ahonen's algorithm can be improved upon by using different correlation functions.

The correlation functions S25, S16, and S28 satisfied the four desirable properties, but S29 did not satisfy these properties. In particular, the correlation function S25 and S28 both were included in the top 10 in simple match. These results indicate that correlation functions S25 and S28 had higher matching rates than the other correlation functions.

7.2.2 Comparing with human subjects

To see the average # of matches among human subjects is interesting and also provides insight into interpreting the average # of matching by the algorithm. For instance, if an algorithm matches 1 on average and the human matches 7, then the performance of the algorithm is almost negligible no matter how much higher its performance is compared to others.

We presented the best, worst and average of human results in Table 5. The results told us that only 1.3 phrases out of 10 picked by a human subject matched with the phrases picked by the others in average. We also conducted a t-Test with the human average and VPF-1 with S25. the human subjects' average was statistically significantly better than the best result obtained by the algorithm with a 95% confidence interval (P=0.02). It would be interesting to see if the worst case of human matching was higher than the algorithm's. The answer was no. It was not statistically significantly better than the machine's. This result indicates that human matching is better than the matching of algorithms in general but not always.

7.2.3 The maximum cases

While reading the above results a question may arise "what will be the best match that the algorithms (VPF-1, VPF-2, and Ahonen) can achieve?" In order to answer this question we picked the best result created by each algorithm. In other words, we picked different correlation functions for the different articles that yielded the best result.

We presented the maximum match cases in Table 6. If we can somehow choose the best correlation function for different article, VPF-1 would achieve a 1.16 match, VPF-2 achieves a 1.23 match, and Ahonen's achieves a 0.9 match. Even using the highest computation among those algorithms examined, Ahonen's algorithm did not perform well. We assume the filtering stage of Ahonen's algorithm filtered many meaningful phrases out or their weighting scheme using the length of a phrase and tightness (refer to [1]) distracted the correlation value of a phrase. Notice that VPF-2 could achieve higher match results if it chooses the best correlation function for each different article. Does this mean VPF-2 finds more meaningful phrases than VPF-1? But didn't VPF-1 score the 3 most exact matches (Table 3)? We cannot resolve this dilemma. But, this result indicates that we can still devise a method of higher matching using more information.

7.3 Analysis of Simple Match

Simple match uses a list of meaningful phrases by taking the union of phrases selected by the 10 human subjects. The phrases found by any method and the phrases in the list are compared. Simple match is not directly related to finding more meaningful phrases, because more meaningful is related to more popular. However, this type of count removed the popularity information.

7.3.1 Top 10 best methods

We ranked all methods in the order of the average and presented top 10 in Table 7. Ahonen's methods with and without gap are also listed for comparison. The results showed VPF-1 produced 8 of the top 10 this time. VPF-2 is ranked 10. Ahonen without gap ranked 63 and Ahonen with gap ranked 153 out of 160. These results also told us that VPF-1 and VPF-2 found more phrases than Ahonen's. The reason for the dominance of VPF-1 in this experiment was that VPF-1 found more less-meaningful phrases than VPF-2, which means, "quantity before quality". Three facts supported this conclusion: the dominance of VPF-1 in simple match, the competitive results of VPF-2 in exact match, and the higher average of VPF-2 in maximum case in Table 6. We also compared the correlation functions, which ranked high to see if they satisfy the 4 correlation properties. Correlation functions S28, S25, S27, S21 and S4 satisfied the correlation properties, but correlation functions S13, S8, S24 and S29 did not. We could not find any clear proof from this table that the desirable properties are related to the performance. VPF-1 with S1 had a higher match percentage than Ahonen's with S32 on average, but the results were not statistically significant.

Table 7. Ranked by average – simple match

Rank	Method	Avg.	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
1	vpf-1 S28	**3.70**	5	4	3	2	4	4	2	3	6	3
2	vpf-1 S25	3.69	5	5	3	2	4	4	2	3	6	3
3	vpf-1 S13	3.67	4	4	3	2	5	4	3	5	4	3
4	vpf-1 S8	3.66	4	5	4	2	4	4	3	3	4	3
5	vpf-1 S27	3.57	4	5	4	2	4	4	3	2	5	3
5	vpf-1 S24	3.57	4	5	4	2	4	4	3	3	4	3
7	vpf-1 S21	3.38	5	5	3	2	5	3	2	3	3	3
8	chans S4	3.36	4	6	3	1	4	6	2	3	3	2
8	vpf-2 S4	3.36	4	6	3	1	4	6	2	3	3	2
10	vpf-1 S29	3.34	4	6	3	2	4	4	1	2	5	3
61	ahonen S32	**2.93**	4	5	2	1	3	3	2	3	5	2
149	ahonen_gap_S32	1.75	3	4	2	1	2	2	0	0	4	0

Table 8. Ahonen with other corr. function-simple match

Rank	Method	Avg.	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
23	ahonen S10	3.20	4	5	2	1	3	3	3	3	5	4
24	ahonen S11	3.20	4	5	2	1	3	3	3	3	5	4
25	ahonen S12	3.20	4	5	2	1	3	3	3	3	5	4
26	ahonen S17	3.20	4	5	2	1	3	3	3	3	5	4
27	ahonen S26	3.20	4	5	2	1	3	3	3	3	5	4
30	ahonen S6	3.18	5	5	3	1	4	3	3	3	5	2
43	ahonen S2	3.02	5	4	2	1	4	3	3	3	4	2
44	ahonen S20	3.02	4	5	2	1	3	3	3	3	5	2
45	ahonen S22	3.02	4	5	2	1	3	3	3	3	5	2
46	ahonen S23	3.02	4	5	2	1	3	3	3	3	5	2

Table 9. Comparing with human – simple match

Method	Avg.	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
Human best	6.3	8.0	5.0	5.0	3.5	6.0	8.9	6.7	7.1	6.7	6.0
Human worst	4.7	5.0	4.4	5.6	3.0	5.0	5.0	4.0	2.0	6.7	6.1
Human avg.	5.6	6.8	4.9	4.2	4.3	4.2	6.7	6.5	6.0	6.8	5.8

Table 10. Maximum cases – simple match

Method	Avg.	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10
VPF-1	4.9	6.0	7.0	4.0	4.0	5.6	4.7	4.0	5.0	6.0	3.0
VPF-2	4.9	6.0	7.0	4.0	4.0	6.0	5.6	4.4	4.0	5.0	3.0
Ahonen	3.6	4.7	4.6	2.7	1.3	5.0	2.7	2.5	3.3	6.0	3.6

Ahonen's algorithm can be improved by incorporating different correlation functions. The correlation functions S6, S10, S11, S12, S17, S26, and S20 appeared in both the exact match and simple match experiments. However, correlation functions S2, S22, and S23 were higher only in the simple match experiment. Therefore, we can say Ahonen's algorithm can be improved by using the correlation functions S6, S10, S11, S12, S17, S26, and S20.

7.3.2 Comparing with human subjects

Human subjects may have different lists than the machine, because there are 9 other people selecting phrases. In order to compensate for this problem, we divided the human's results by 9 not 10. The best, worst, and average human matches are listed inTable 9. VPF-1 with S1 (0.49) scores higher than the worst case of human matching (0.47). This result is different than the results of exact match (VPF-1 with S25 was lower than the worst case in exact match by human subjects). This implies that human subjects share only some popular phrases, while the other phrases selections vary.

7.3.3 The maximum cases

We also listed the maximum cases for each article in Table 10. VPF-1 and VPF-2 yielded similar results – both 0.49 on average, but the value details were different. And both VPF-1 and VPF-2 had higher average match values than Ahonen's algorithm did. These results also support the case our algorithms find more matching phrases in the maximum case.

8. CONCLUDING REMARKS

The goal of this paper is to devise an algorithm, which finds more meaningful phrases than older methods. We proposed two algorithms, which meet this goal: VPF-1 and VPF-2. We also coordinated these algorithms with 32 different correlation functions. They regenerate sequences recursively with the words selected in the previous stage and search for increased length of phrases in time O(N), where N is the page size. Since our algorithm uses average as a threshold and stops when the length of phrases does not increase, no user-defined parameter is required. By comparing the top 10 best measures and comparing the average of best cases, we observed that when we add pruning, both algorithms (VPFs) had improved performance.

In order to choose the best method, we conducted an experiment by asking 10 human subjects to select 10 phrases from 10 different articles. We compared the # of matching phrases chosen by a method to those phrases chosen by 10 human subjects. We concluded that VPF-1 with S25 found a statistically significantly greater number of meaningful phrases than Ahonen's previous method. We suspect the filtering stage of Ahonen's algorithm filtered many meaningful phrases out or their weighting scheme using the length of a phrase and tightness (refer to [1]) distracted the correlation value of a phrase.

Interestingly, the correlation functions S25 and S28 were both included in the top 10 in both exact match and simple match. This result indicates the correlation functions S25 and S28 had higher matching rates than the other correlation functions. These two correlation functions both satisfied Piatetsky-Shapiro's three desirable properties [15] and are able to distinguish positive and negative correlations [17]. We can also improve Ahonen's algorithm by incorporating correlation functions S10, S11, S12, S17, S26, S6, and S20. Those functions resulted in a higher match of average scores for both exact match and simple match experiments.

The performance of our two methods varied depending on the articles selected. We currently do not understand the reason for the variance in performance over different articles. We assume it is due to the intrinsic characteristics of an article. In the future, we hope to apply these collected phrases to improve a recent hierarchical user-interest model [11]. We will also investigate the property of an article that keeps the size of the next sequence less than $S \log S$, where S is the size of the current sequence.

Table 11. Correlation functions

	Name	Formula
1	ϕ-coefficient (CE)	(AB - (A × B)) / Sqr(A × B × (1 - A) × (1 - B))
2	Goodman-Kruskal's	(MAX(AB, AB′) + MAX(A′B, A′B′) + MAX(AB, A′B) + MAX(AB′, A′B′) - MAX(A, A′) - MAX(B, B′)) / (2 - MAX(A, A′) - MAX(B, B′))
3	Odds ratio (OR)	D((AB × A′B′), (AB′ × A′B))
4	Yule's Q (YQ)	(AB × A′B′ - AB′ × A′B) / (AB × A′B′ + AB′ × A′B)
5	Yule's Y (YY)	(Sqr(AB × A′B′) - Sqr(AB′ × A′B)) / (Sqr(AB × A′B′) + Sqr(AB′ × A′B))
6	Kappa (k) (KP)	(AB + A′B′ - (A × B) - (A′ × B′)) / (1 - (A × B) - (A′ × B′))
7	Mutual Information (M)	(AB × log2(AB / (A × B)) + AB′ × log2(AB′ / (A × B′)) + A′B × log2(A′B / (A′ × B)) + A′B′ × log2(A′B′ / (A′ × B′))) / (MIN(-(A × log2(A) + A′ × log2(A′)), -(B × log2(B) + B′ × log2(B′))))
8	J-Measure	MAX(AB × log2(P(B|A) / B) + AB′ × log2(P(B′|A) / B′), AB × log2(P(A|B) / A) + A′B × log2(P(A′|B) / A′))
9	Gini index (G)	MAX(A(pow(P(B|A),2) + pow(P(B′|A),2)) + A′(pow(P(B|A′),2) + pow(P(B′|A′),2)) - pow(B,2) - pow(B′,2), B(pow(P(A|B), 2) + pow(P(A′|B), 2)) + B′(pow(P(A|B′), 2) + pow(P(A′|B′), 2)) - pow(A, 2) - pow(A′, 2))
10	Support	AB
11	Confidence (c)	MAX(P(B|A), P(A|B))
12	Laplace (L)	MAX((100 × AB + 1) / (100 × A + 2), (100 × AB + 1) / (100 × B + 2))
13	Conviction (CV)	MAX((A × B′) / AB′, (B × A′) / A′B)
14	Interest (IT)	AB / (A × B)
15	Cosine (IS)	AB / Sqr(A × B)
16	Piatetsky-Shapiro's (PS)	AB - A × B
17	Certainty Factor (CF)	MAX((P(B|A) - B) / (1 - B), (P(A|B) - A) / (1 - A))
18	Added Value (AV)	MAX(P(B|A) - B, P(A|B) – A)
19	Collective strength (S)	((AB + A′B′) / (A × B + A′ × B′)) × ((1 - A × B - A′ × B′) / (1 - AB - A′B′))
20	Jaccard	AB / (A + B - AB)
21	Klosgen (KL)	Sqr(AB) × MAX(P(B|A) - B, P(A|B) - A)
22	MI	Log2(AB / (A × B))
23	STC_MIN	MIN(P(B|A), P(A|B))
24	EMI	AB × log(AB / (A × B)) + AB′ × log(AB′ / (A × B′)) + A′B × log(A′B / (A′ × B)) + A′B′ × log(A′B′ / (A′ × B′))
25	AEMI	AB × log(AB/ A × B) - AB′ × log(AB′/ A × B′) - A′B × log(A′B/ A′ × B) + A′B′ × log(A′B′/ A′ × B′)
26	dMAX	AB × MAX(P(B|A), P(A|B))
27	dMI	AB × log2(AB / (A × B))
28	AEMI3	AB × log(AB/ A × B) - AB′ × log(AB′/ A × B′) - A′B × log(A′B/ A′ × B)
29	dMIN	AB × MIN(P(B|A), P(A|B))
30	dMIN2	1 + AB × log(MIN(P(B|A), P(A|B)))
31	NegativeCosine	(1 - AB) / Sqr((1 - A) × (1 - B))
32	MutualConfidence	(AB / A + AB / B) / 2

9. ACKNOWLEDGMENTS

We thank all those who voluntarily joined our experiment and Matt Scripter's valuable comments.

10. REFERENCES

1. Ahonen, H., Heinonen, O., Klemettinen, M., and Verkamo, A.I. Applying Data Mining Techniques for Descriptive Phrase Extraction in Digital Document Collections, Proceedings of the Advances in Digital Libraries Conference, 1998

2. Chan, P.K. A non-invasive learning approach to building web user profiles, KDD-99 Workshop on Web Usage Analysis and User Profiling, 7-12, 1999.

3. Chen, L., and Sycara, K. Webmate: A personal agent for browsing and searching, In Proc. 2nd Intl. Conf. Autonomous Agents, pp. 132-139, 1998.

4. Croft, W.B. (editor), Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval, Massachusetts, Kluwer Academic Publishers, pp. 243, 2000.

5. Croft, W.B., Turtle, H.R. and Lewis, D.D. The use of phrases and structured queries in information retrieval, ACM SIGIR, pp. 32-45, 1991.

6. Fagan, J.L. Automatic phrase indexing for document retrieval, Proceedings of the Tenth Annual ACM SIGIR Conference on Research & Development in Information Retrieval, pp. 91-101, 1987.

7. Frakes, W.B., and Baeza-Yates, R. Information Retrieval: Data Structures and Algorithms, Prentice-Hall, 1992.

8. Gokcay, D. and Gokcay, E. Generating titles for paragraphs using statistically extracted keywords and phrases, Systems, Man and Cybernetics, 1995. 'Intelligent Systems for the 21st Century'., IEEE International Conference on, Volume: 4 , 22-25 Oct. 1995

9. Hilderman, R. and Hamilton, H. Evaluation of interestingness measures for ranking discovered knowledge. In Proc. Of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2001.

10. Kamber, M. and Shinghal, R. Evaluating the interestingness of characteristic rules. In Proc. Of the Second Int'l Conference on Knowledge Discovery and Data Mining, Pages 263-266, Portland, Oregon, 1996.

11. Kim, H. and Chan, P.K. Learning implicit user interest hierarchy for context in personalization. International Conference on Intelligent User Interfaces, 101-108, 2003.

12. Lima, E.F. and Pedersen J.O. Phrase Recognition and expansion for short, precision-biased queries based on a query log, SIGIR, 1999.

13. Mannila, H. and Toivonen, H. Discovering generalized episodes using minimal occurrences, Knowledge Discovery and Data Mining, 1996

14. Mitra, M., Buckley, C., Singhal, A. and Cardie, C. An Analysis of Statistical and Syntactic Phrases, Proceedings of RIAO-97, 5th International Conference, 1997.

15. Piatetsky-Shapiro, G. Discovery, analysis and presentation of strong rules. In G. Piatetsky-Shapiro and W. Frawley, editors, Knowledge Discovery in Database, pages 2299-248. MIT Press, Cambridge, MA, 1991.

16. Rosenfeld, R. Adaptive Statistical Language Modeling: A Maximum Entropy Approach, PhD thesis, Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1994.

17. Tan, P. and Kumar, V. Interestingness Measure for Association Patterns: A Perspective*, KDD, 2000.

18. Tan, P. Kumar, V. and Srivastava J. Selecting the Right Interestingness Measure for Association Patterns. In Porc. ACM SIGKDD, 2002.

19. Turpin, A. and Moffat, A. Statistical phrases for vector-space information retrieval. In Proceedings of SIGIR-1999, pp. 309--310, 1999.

20. Wu, H. and Gunopulos, D. Evaluating the Utility of Statistical Phrases and Latent Semantic Indexing for Text Classification, IEEE International Conference on Data Mining, 2002.

21. Zamir, O., and Etzioni, O. Web document clustering: a feasibility demonstration. In Proc. SIGIR-98, 1998.