Identifying Variable-Length Meaningful Phrases with Correlation Functions

Hyoung-rae Kim and Philip K. Chan
Department of Computer Sciences, Florida Institute of Technology
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 a 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.

1. Introduction

Statistical phrase finding algorithms are mainly used for improving the performance of information retrieval [6,7,12,21]. There are three main approaches: syntactic [13], statistical [15], and hybridized [9]. 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 [21,22], 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 [23], 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 the generating stage and a threshold for the maximum phrase length – are too strict. Zamir and Etzioni [23] 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 a variable-length phrase finding algorithm (VPF), which finds more meaningful variable-length phrases. VPF is designed to remove the maximum length of phrases in Ahonen's algorithm, and fix the problem in Chan's algorithm – the details will be explained in next section. VPF adds correlated words one by one to the phrases made in the previous stage, maintaining high correlation within a phrase. VPF also applies pruning to remove less meaningful phrases at the end. VPF not only show higher performance but also is more robust and has lower time complexity than Chan's algorithm. The main contributions are:

- We proposed a variable-length phrase-finding algorithm (VPF) that is designed for finding meaningful phrases;
- The time complexity remains as O(N) where N is the size of the input sequence under a specified condition;
- This algorithm does not need any user-specified parameters such as a phrase length;
- The algorithm achieves further improved performance by pruning less meaningful phrases;
- More meaningful phrases than previous methods are found and the improvement in performance is statistically significant.

The rest of this paper is as follows: Section 2 presents related work regarding statistical phrase-finding methods; Section 3 provides detailed description of our variable-length phrase-finding algorithm (VPF) and describes the desirable properties of correlation functions and lists all correlation functions we used; Section 4 discusses about experiment; Section 5 presents and analyzes our results; Section 6 summarizes our 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 [22] 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 [7] selected phrases having a document frequency of at least 55 and a high co-occurrence in the same sentence. Mitra et al. [15] collected all pairs of non-function words that occur contiguously in at least 25 documents. Turpin and Moffat [21] 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, this method could remove meaningful phrases in a document. Furthermore, only two-word phrases are considered, whereas ours has no limitation in phrase length.

Croft, et al. [6] 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* (Term Frequency Inverse Document Frequency) 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, 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 use simple phrase separators (i.e., stop-words and non-alphabet characters), which generalizes our method independent from a certain language.

Gokcay and Gokcay [9] 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 the 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's method [1] finds all possible combinations of words within a fixed window using Mannila and Toivonen's [14] algorithm. 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",… "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. They also allowed gaps within a phrase. Even with the algorithm's exhaustive examination, its performance is, as will be shown later, lower than our method.

Zamir's algorithm [23] uses only frequency information. They collect neither too frequent nor too rare phrases. 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. However, our method does not need any parameters.

Chan's phrase-finding algorithm [2] calculates the correlation values of all pairs. 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 values higher than the threshold, Chan's algorithm will generate a word "abbbb" that does not exist in the string. 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. Chan [2] implemented the algorithm in time $O(D^2)$, where D is the number of distinct words, while our implementation consumes $O(N)$, where N is the size of an input sequence.

3. Variable-length phrases

Our algorithm consists of two components: the main algorithm and the correlation function. In preparation for VPF, we extract words from a web page visited by the user, filter them through a stop list, and stem them [1,2,8,23]. Other phrase-finding algorithms also require these pre-processing steps.

3.1. VPF Algorithm

Our algorithm adds words one by one to the phrases found in a previous stage – each stage corresponds to each recursion in the VPF algorithm. 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.

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). It uses four main variables – *List*, *SEQ*, *Corr, and Fitness*. The *List* variable stores all collected phrases in a *Hash* attribute. Each element (phrase) of the *Hash* attribute keeps correlation value in *sim* attribute and the position list in *posi* attribute. VPF first makes a linked list (*SEQ*) with an input example. Each word and word position are stored in each node in the linked list. Then, all 1-gram distinct words are stored in *List*[1].*Hash*. *Corr* is a chosen correlation function and *Fitness* is a function that measures whether a phrase is valid or not. The *List* , *SEQ*, *Corr* and *Fitness* are passed to *BeSpecific* procedure and this finds all phrases. Once the phrases are acquired, they are pruned. The pruning process simply removes all sub-phrases that have a *sim* value lower than the *sim* value of the super-phrases.

The *BeSpecific* procedure receives six parameters: *List* that stores all phrases; *L* which is the length of phrases (initial value is 2); *thre* which keeps the calculated threshold value differentiating "strong" relations from "weak" relations (initialized to 0); *SEQ* which stores the linked list; *Corr* and *Fitness*. *BeSpecific* recursively creates new sequences by removing nodes that are not in the *Hash* table generated in the previous stage, and also by removing consecutive nodes whose lengths are shorter than *L*. Since it removes words that are not in the *Hash* table generated in the previous stage, there can be gaps between nodes. Once the new sequence is generated, it collects all *L*-grams having no gaps from the sequence. The threshold is calculated when *L*=2 only once by averaging all correlation values between the first and second word in each element in *List*[2].*Hash*. The *for* loop removes any element with low correlation values (*sim*) from *Hash*. The *sim* property of *p* keeps the correlation between a sub phrase, *p*[1..*L*-1], and the last word, *p*[*L*], where *p* is a phrase consisting of *L* words. For example, if *p*="computer science seminar", then *p*[1..*n*-1]="computer science" and *p*[*n*]="seminar". The *Intersection* counts all adjacent points based on the distance. The intersection between the positions of *p*[1..*L*-1] and *p*[*L*] becomes the positions of *p*. The *BeSpecific* procedure recursively increases the phrase length *L* until *Hash* become empty. We can also apply various correlation functions in the place of *Corr*. The *Fitness* function gets the frequency of a phrase as an input and returns *true* or *false*. We use a simple *Fitness* function that returns *false* if the input is 0 and *true* otherwise.

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 his one chance to come before any other words, so we adjust the pre-percentage of the word by deducting one from the number of occurrences of the word. The post-percentage is vice versa. 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:

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

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

```
Input: Example- a sequence of words
Output: Collected phrases
Function VPF (Example) return phrases
 SEQ- linked list, each node has a word and a
     position.
 List- array of Hash table, each word in a Hash
     has sim and posi attributes.
 Corr- a correlation function
 Fitness- a function that measures whether a
     phrase is valid or not
1.  SEQ←linked list made by the input Example
2.  List[1].Hash←store all 1-grams, each word
     keeps its positions
3.  BeSpecific(List, 2, 0, SEQ, Corr, Fitness)
4.  Prune phrases in the List
5.  Return all phrases in the List
End Function

Input: List- store array of Hash table
 L- length of phrase, initialized to 2
 thre- threshold value, initialized to 0
 SEQ- linked list, the size reduces every
     iteration
BeSpecific(List, L, thre, SEQ, Corr, Fitness)
1.  if List[L-1].Hash is empty then stop
2.  SEQ←remove nodes that are not in the List[L-
       1].Hash and also remove consecutive nodes
       which length is shorter than L
3.  List[L].Hash←Collect all L-grams from SEQ
4.  if L=2 then thre←Average correlation across
       all words in List[2].Hash
5.  for each p in List[L].Hash do
6.    A ← pre-percentage of p[1..L-1]
7.    B ← post-percentage of p[L]
8.    p.sim←Corr(A,B,A∩B)
9.    remove any p for which p.sim is lower then
       thre
10.   p.posi←Intersection of p[1..L-1].posi and
       p[n].posi with distance of L-2
11.   remove any p for which Fitness(p.posi)
       does not satisfy
12. BeSpecific(List, L+1, thre, SEQ, Corr,
       Fitness)
End Procedure
```

Figure 1. VPF algorithm

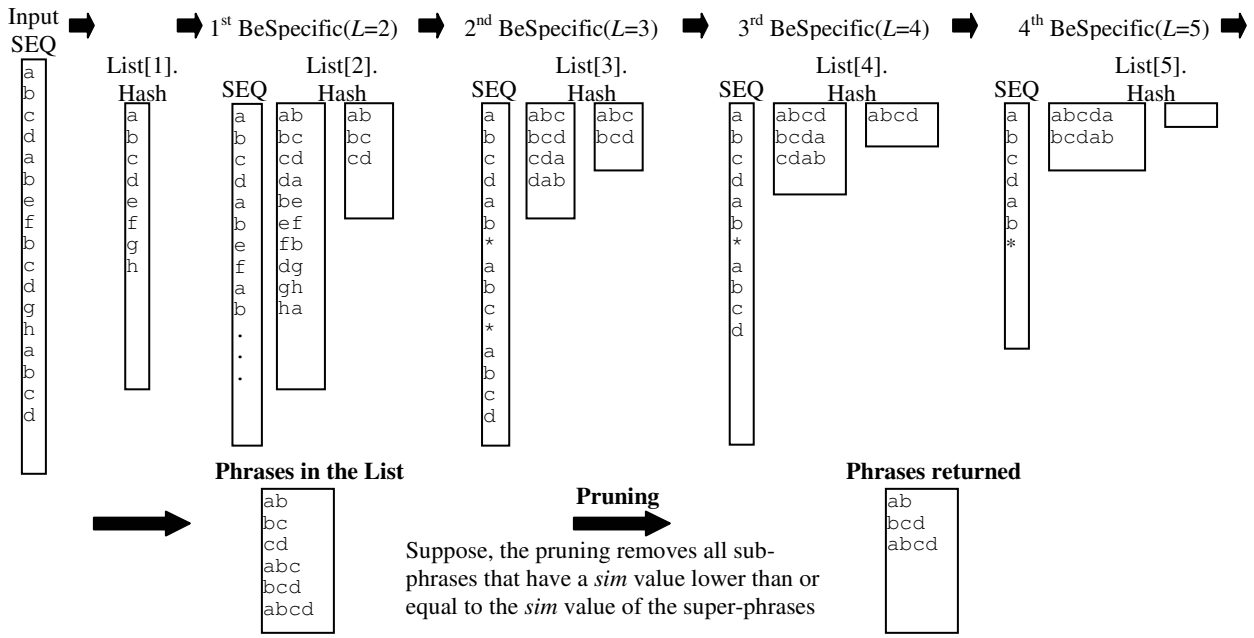

Figure 2. Example of VPF

The most expensive part in the *BeSpecific* procedure will be scanning a sequence when *L* is 2. Even though the *BeSpecific* is called *L* times, as *L* increases the size of the sequence decreases drastically. The *Corr* function has O(*T*) time complexity, where *T* is the position length. But, as T increases the number of element in a Hash decrease. We, therefore, can claim that the time complexity of VPF in general case is roughly: O(*S*), where *S* is the sequence size.

For example, in Figure 2 we have an input string *S* = "abcdabefbcdghabcd," in which each letter represents a word. The correlation function simply returns the frequency of a phrase using a threshold value of 1.5. *List*[1].*Hash* contains all distinct words. *List*[2].*Hash* originally contains all possible 2-word phrases, then removes any phrase that occur less than 2 times, resulting in {"ab", "bc", "cd"}. We remove all words that are not in the *List*[2].*Hash* from the 1st *SEQ* resulting in 2nd *SEQ*. *List*[3].*Hash* contains all 3-word phrases, and then remove "cda" and "dab" because they occur only once. When we came to *BeSpecific* for the 3rd time, we removed "*abc*" from the 3rd *SEQ*, because their consecutive length is less than 4 – the size of L increases every time we come to *BeSpecific*. When we run the 4th *BeSpecific*, we can remove all phrases ("abcda" and "bcdab") in *List*[5].*Hash*, because they occur only once. Since the 5th *Hash* is empty, the *BeSpecific* stops. After the *BeSpecific*, the *List* keeps {"ab", "bc", "cd", "abc", "bcd", "abcd"}. Suppose the pruning removes all sub-phrases that have a *sim* value lower than or equal to the *sim* value of the super-phrases. The occurrences of phrases are: "ab"-3, "bc"-3, "cd"-3, "abc"-2, "bcd"-3, and "abcd"-2. We remove "bc" and "cd" because "abc" subsumes them and has equal frequency. "abc" is removed by "abcd" with the same reason. The final returned phrases are {"ab", "bcd", "abcd"}.

3.2. Correlation Functions

The VPF algorithms build phrases; and correlation functions actually calculate the weight of a phrase. The correlation functions are 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 [3,17]. All correlation measures are not equally good at capturing the dependencies between different events [20]. It is because each correlation function biases toward different individual event probabilities and joint probabilities. Piatetsky-Shapiro [16] has proposed three key properties that a good correlation function, *F*, for events *A* and *B* 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 is independent when its determinant is equal to zero [19]. Another

important operation in finding phrases is distinguishing between positive and negative correlations (*P*4). Measuring their *cross product ratio* (*CPR*) can assess the significance of the correlation between *A* and *B* [17] 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. *P*4 is that *F* can distinguish positive and negative correlation of *A* and *B*. 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. [20] illustrated those properties and extended them to each of the existing measures to determine if the existing measure satisfies the properties required [11]. Some of these properties have been extensively investigated in the data mining literature [10,19,20]. 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 6 in Appendix A.

4. Experiments

4.1. Evaluation Data and Procedures

We use five New York Times articles and five Web pages collected from our department server. We use 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 and we asked 10 human subjects, other than the authors, to read the 10 articles. Each article contains about 1,300 words - 720 words after removing stop-words. Since we assigned 6 as a threshold of maximum phrases length for Ahonen's and Chan's, the total possible number of phrases for each article is approximately 3600 (=720×5). Of the 10 subjects, 4 were graduate students from a department of computer sciences and 6 were undergraduates with various majors.

We asked the 10 human subjects to choose their top 10 meaningful phrases for each article or Web page. One might insist that the results will be different depending on how 10 humans are chosen. If all volunteers have the same background the matching rate will be higher than the normal case. However, since we are not comparing the algorithm with humans, but comparing among algorithms, it does not matter how we chose the 10 volunteers. Furthermore, since the algorithm finds phrases statistically that cover general human meaningfulness, we choose 10 human subjects arbitrarily.

The instruction that we gave them were:

```
Identify the 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 also count average matching of humans – in this case, we divided the sum by 9. There are cases for the human or algorithm to select less than 10 phrases. In order to be fair in these cases, we use an additional adjustment function. We also attempt to prevent a measure from being scored 1 by finding one phrase and having one matched phrase by chance – the results are too sensitive. We, therefore, decided to give a lower incentive as a measure finds fewer phrases 20% at the most. For example, if there are 5 matches out of 10, the number 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 generalized formula is:

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

, where *m* is the number of matched words and *f* is the number of selected words.

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 correlation functions: conditional probability (Confidence, F11) for filtering phrases and mutual confidence (F32) 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.

4.2. Evaluation Criteria

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*. We have 128 methods (4 algorithms × 32 correlation functions) – the 4 algorithms are VPF, Chan's, Ahonen's, and Ahonen's with gap, and the correlation functions F1 through F32 are in Appendix A.

Table 1. With pruning vs. without pruning

Rank	Method	Avg. across humans and articles		Ratio of
		With-pruning	Without-pruning	Improv.
1	VPF_F25	**0.933**	0.883	5.7%
2	VPF_F16	**0.920**	0.866	6.2%
3	VPF_F28	**0.912**	0.871	4.7%
4	VPF_F8	**0.824**	0.707	16.5%
5	VPF_F10	0.819	**0.825**	-0.7%
6	VPF_F29	**0.814**	0.810	0.5%
7	VPF_F27	**0.812**	0.796	2.0%
8	VPF_F24	**0.812**	0.758	7.1%
9	VPF_F13	**0.772**	0.666	15.9%
10	VPF_F26	**0.726**	0.683	6.3%

The number of *exact matches* of a method is measured by the percentage of the matches between the human's and a method's. We count each match with a human's and then average the 10 compared results. This counting approach assigns more weights to the more meaningful words – more meaningful word means that they were selected by more human subjects. If a phrase is selected by several human subjects, every match is counted. Therefore, finding more popular phrases increases the matching average. The number of matches will be very low, because only 10 phrases selected by a method and a human respectively are going to be compared.

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. *Simple match* is not directly related to finding more meaningful phrases, because this count removes the popularity information. We count this only to support the result of the *exact match*.

Comparison of the results is done using a matched-pair design [18]. In this design, the top ten phrases in the ranking generated are compared. The comparison, which simply identifies if one group of ten phrases is better than the other, is on the basis of precision in other words the number of matched phrases. This type of evaluation has the following advantages: It is realistic in the sense that many information retrieval systems are interested in the top group. Traditional recall/precision tables are very sensitive to the initial rank positions and evaluate entire rankings [5]. Another advantage is that significance measures can be readily used.

5. Results and analysis

Before comparing our algorithm with existing methods we need to decide whether to use pruning or not. After that we will be able to perform the comparison. In evaluating our method against related algorithms we use different scoring methods: exact match and simple match.

5.1. With-pruning vs. Without pruning

The VPF algorithm has a pruning function. The results differed whether we used the pruning function or not. We compared them by comparing the top 10 best methods with *exact match* (Sec 6.2). By composing the algorithm and 32 correlation functions (in Appendix A), we generated 32 methods. We ranked the top 10 methods using "with pruning" and presented the corresponding results of "without pruning" next in Table 1. The values are the average of matches across 10 human subjects and 10 articles. Most methods yielded improved results when they had been pruned. The top method VPF with F25 had improved its performance by 5.7%. With pruning is statistically significantly better than without pruning with a 95% confidence interval (P=0.004).

5.2. Analysis with Exact Match

Because "with-pruning" achieves a higher matching rate than "without-pruning", we decided to use pruning in our algorithms for the rest of our experiment.

5.2.1. Top 10 methods. The main purpose of the analysis in this section is to choose the best method. Which method is the best is the most interesting question. We averaged the results from 10 articles and 10 human subjects and sorted by the average to rank all 128 methods. We presented the results in Table 2 and included the rank, methods used, and the average. Each method was composed of an algorithm and a correlation function. Notice that, we also presented the results of previous methods. Ahonen used correlation function F32. 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 and correlation functions F25 followed by F16 and F28 – all those three correlation functions satisfied Piatetsky-Shapiro's three desirable properties and distinguish positive from negative correlations. The best method VPF with F25 matched 0.93 phrases on average with the phrases selected by a human subject. In the next section we measured the average number of matching phrases between human subjects and compared those results to the results from methods.

Interestingly, VPF won the top 3. Chan's algorithm occupied the next ranks. Another observation was that the correlation functions F25, F16, and F28 that marked high rank with VPF also marked high rank with Chan's. This observation implied that the performance also depends on the correlation functions.

Table 2. Ranked by average across humans and articles – Exact match

Rank	Method	Avg.	Rank	Method	Avg.
1	vpf_F25	0.933	**13**	ahonen_F6	0.797
2	vpf_F16	0.920	**15**	ahonen_F10	0.779
3	vpf_F28	0.912	**15**	ahonen_F11	0.779
4	chans_F16	0.858	**15**	ahonen_F12	0.779
5	chans_F25	0.856	**15**	ahonen_F17	0.779
6	chans_F29	0.850	**15**	ahonen_F26	0.779
7	chans_F28	0.848	**20**	ahonen_F20	0.774
8	vpf_F8	0.824	**20**	ahonen_F23	0.774
9	vpf_F10	0.819	**24**	ahonen_F32	0.767
10	vpf_F29	0.814	**105**	ahonen_gap_F32	0.452

Table 3. Exact match across humans

	Avg. across 10 articles
Human best	1.48
Human avg.	1.30
Human worst	1.03

Table 4. Ranked by average across humans and articles – Simple match

Rank	Method	Avg.	Rank	Method	Avg.
1	vpf_F28	3.696	**12**	ahonen_F10	3.195
2	vpf_F25	3.689	**12**	ahonen_F11	3.195
3	vpf_F13	3.672	**12**	ahonen_F12	3.195
4	vpf_F8	3.656	**12**	ahonen_F17	3.195
5	vpf_F27	3.575	**12**	ahonen_F26	3.195
5	vpf_F24	3.575	**17**	ahonen_F6	3.181
7	vpf_F21	3.377	**23**	ahonen_F2	3.025
8	vpf_F29	3.342	**24**	ahonen_F20	3.018
9	vpf_F16	3.321	**24**	ahonen_F22	3.018
10	chans_F29	3.282	**24**	ahonen_F23	3.018
22	chans_F25	3.053	**33**	ahonen_F32	2.934
			118	ahonen_gap_F32	1.755

Table 5. Simple match across humans

	Avg. across 10 articles
Human best	6.3
Human avg.	5.6
Human worst	4.7

Unfortunately, Ahonen's algorithm ranked 24 and Ahonen_gap 105. These methods matched 0.767 and 0.452 numbers of phrases with human subjects respectively. The low performance with gap is the same phenomenon as shown in [1]. We conducted t-Test (paired two sample for means) between VPF with F25 and Ahonen with F32. There was a clear statistically significant difference between the two methods with 95% confidence (P=0.016). Therefore, we can conclude that VPF with F25 found statistically significantly more meaningful phrases than Ahonen's previous algorithm.

Ahonen's algorithm with other correlation functions received higher ranks such as F6, F10, F11, F12, F17, F26, and F20 as shown in Table 2. They all ranked 13, 15, and 20, which are higher than Ahonen's original method (24). This indicates Ahonen's algorithm can be improved upon by using different correlation functions.

5.2.2. Comparing with human subjects. To see the average number of matches among human subjects is interesting and also provides insight into interpreting the average number 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, average, and worst matching human results in Table 3. 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 on average. This is not an unrealistic result. Considering that each document has approximately 1,300 words, more than 7779 possible phrase combinations exist and each person has a different background, matching 1.3 phrases out of 10 on average is a extraordinarily reasonable result. Our method achieved a result (0.93), which was close to the typical human result. We also conducted a t-Test with the human average and VPF with F25. 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.

5.3. Analysis with Simple Match

This *simple match* count showed similar ranking to the *exact match*. VPF with F28 followed by F25 and F13 had the top matching rates: 3.70, 3.69, and 3.67 respectively as shown in Table 4. Since *simple match* uses a list of meaningful phrases by taking the union of phrases selected by the 10 human subjects, average number of matching phrases is higher than the average by *exact match*. Chan's with F25 ranked 22 (3.05 matching rate), Ahonen without gap ranked 33 (2.93), and Ahonen with gap ranked 118 (1.76) out of 128. Chan's original method ranked 22 (3.053). These results also told us VPF found more phrases than Ahonen's and Chan's. The result from *simple match* also indicated that Ahonen's algorithm could be improved by incorporating different correlation functions.

We also attempted to compare the methods' results with the results from humans. Human matched the list 5.6 out of 10 on average; the best and worst cases are 6.3 and

4.7 as shown in Table 5. The result 3.69 of method VPF with F25, which was the highest score with exact match, was quite significant considering that we only used the statistical information.

6. Conclusions

We proposed a variable-length phrase-finding algorithm, which find more meaningful phrases – VPF – than older methods – Ahonen's and Chan's algorithms. 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.

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 number of matching phrases chosen by a method to those phrases chosen by 10 human subjects. By comparing the top 10 best measures (matched-pair design [18]), we observed that when we add pruning, the algorithm (VPF) had improved performance.

We concluded that VPF with F25 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 [1] distracted the correlation value of a phrase. Interestingly, the correlation functions F25 and F28 were both included in the top 10 in both *exact match* and *simple match*. This result indicates the correlation functions F25 and F28 had higher matching rates than the other correlation functions. These two correlation functions both satisfied desirable properties for phrases. We can also improve Ahonen's algorithm by incorporating correlation functions F10, F11, F12, F17, F26, F6, and F20. Those functions resulted in a higher match of average scores for both *exact match* and *simple match* experiments.

The performance of our method 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, because the human subjects' results are also different depending on the articles. Phrases in VPF grow backwards; however, in the future we will devise an algorithm that grows both forwards and backwards.

7. References

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

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

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

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

5. W.B. Crotf and R. Das, Experiments with Query Acquisition and Use in Document Retrieval Systems, In Proc. of 13th ACM SIGIR, 1989.

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

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

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

9. D. Gokcay and E. Gokcay, 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

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

11. M. Kamber and R. Shinghal, 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.

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

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

14. H. Mannila and H. Toivonen, Discovering generalized episodes using minimal occurrences, *In Proc. of Knowledge Discovery and Data Mining*, 1996.

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

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

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

18. S.E. Robertson, The methodology of information retrieval experiment, In: Sparck Jones, editor, *Information Retrieval Experiment*, London: Butterworths, 9-31, 1981.

19. P. Tan and V. Kumar, Interestingness Measure for Association Patterns: A Perspective*, *In Proc. of KDD*, 2000.

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

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

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

23. O. Zamir and O. Etzioni, Web document clustering: a feasibility demonstration. *In Proc. of SIGIR*, 1998.

Appendix A

Table 6. 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