Interesting-Phrase Mining for Ad-Hoc Text Analytics

Srikanta Bedathur[†], Klaus Berberich[†], Jens Dittrich[‡],
Nikos Mamoulis[†,*], Gerhard Weikum[†]

[†]Max-Planck-Institut für Informatik
Saarbrücken, Germany

[‡]Saarland University
Saarbrücken, Germany

{bedathur,kberberi,nmamouli,weikum}@mpi-inf.mpg.de
jens.dittrich@cs.uni-saarland.de

ABSTRACT

Large text corpora with news, customer mail and reports, or Web 2.0 contributions offer a great potential for enhancing business-intelligence applications. We propose a framework for performing text analytics on such data in a versatile, efficient, and scalable manner. While much of the prior literature has emphasized mining keywords or tags in blogs or social-tagging communities, we emphasize the analysis of interesting phrases. These include named entities, important quotations, market slogans, and other multi-word phrases that are prominent in a dynamically derived ad-hoc subset of the corpus, e.g., being frequent in the subset but relatively infrequent in the overall corpus. We develop preprocessing and indexing methods for phrases, paired with new search techniques for the top-k most interesting phrases in ad-hoc subsets of the corpus. Our framework is evaluated using a large-scale real-world corpus of New York Times news articles.

1. INTRODUCTION

With the dramatic growth of business-relevant information in various textual sources, such as user-interaction logs (web clicks etc.), news, blogs, and Web 2.0 community data, text analytics is getting a key role in modern data mining and Business-Intelligence (BI) for decision support. Analysts are often interested in examining a set of specifically compiled documents, to identify their characteristic words or phrases or discriminate it from a second set. Tag clouds and evolving taglines are prominent examples of this kind of analyses [2, 6, 18]. While there is ample work on this topic for word or tag granularities, there is very little prior research on mining variable-length phrases. Such interesting phrases include names of people, organizations, or products, but also news headlines, marketing slogans, song lyrics, quotations of politicians or actors, and more.

In this paper, we focus on the analysis of *interesting phrases* in *ad-hoc*, dynamically derived document collections, for example, by a keyword query or metadata-based search from a large document corpus. Interestingness can be defined with the help of statistical measures that compare the *local* frequency of a phrase in the ad-hoc collection with its *global* frequency in the entire archive. For example, consider the results of keyword query "Steve Jobs" on a news archive. The most interesting phrases may include *"apple chief executive", "mac os x", "the computer maker"*. The ratio local/global frequency of these phrases is high, therefore they are deemed appropriate in characterizing the query results.

In [23] a *phrase inverted index* is developed for finding the most interesting phrases in an ad-hoc subset $\mathcal{D}'$ of the overall corpus $\mathcal{D}$. As a preprocessing step, for each phrase, identifiers of documents in $\mathcal{D}$ that contain the phrase are collected into an index list, built in an IR-style inverted-file fashion [28]. In order to compute the frequencies of the phrases in $\mathcal{D}'$, the inverted lists are accessed and intersected with $\mathcal{D}'$. An approximate counting technique that intersects only a sampled subset of each list with $\mathcal{D}'$ is proposed; still, a very large number of lists has to be accessed – potentially as large as the number of phrases, regardless of the size of $\mathcal{D}'$. As news, blogs, and web-usage corpora become rapidly larger, the phrase-inverted-index method becomes practically infeasible for interactive analytics. In fact, the experiments in [23] only reported results on a corpus of 30,000 publications.

In this paper, we develop an efficient alternative to [23] with much better scalability. We pre-process the documents in the entire corpus $\mathcal{D}$ and extract all phrases (above some minimum-support threshold). We then encode and index the phrases contained in each document in a *forward index list*. Given a subset $\mathcal{D}' \subset \mathcal{D}$, in order to determine the frequencies and compute the interestingness of the phrases there, we scan and merge the forward index lists of the documents in $\mathcal{D}'$. We propose several variants of this approach, based on different ways of ordering and compressing the phrases in the lists. These variants in turn lead to alternative algorithms for the phrase mining, with different capabilities for pruning the search space. As the number of phrases that are contained in $\mathcal{D}'$ can be very large, we focus on finding the top-k interesting phrases. We offer a systems-level solution that scales to very large corpora $\mathcal{D}$. Our methods are evaluated using a corpus of nearly two million articles from the New York Times archive. Our problem setting differs from classic sequence mining [27] by the *ad-hoc* nature of the subset $\mathcal{D}'$ of $\mathcal{D}$: $\mathcal{D}'$ is dynamically derived from queries and we gear for this novel situation by judicious indexing of $\mathcal{D}$.

The rest of the paper is organized as follows. Section 2 defines a representative interestingness measure for phrases in an ad-hoc subset of a corpus. In Section 3, we present alternative methods for indexing the document corpus and searching for interesting phrases, including the framework that we propose in this paper. We experimentally demonstrate the efficiency and scalability of our approaches in Section 4. Section 5 reviews related work and Section 6 concludes the paper.

[*]on leave from the University of Hong Kong

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were presented at The 36th International Conference on Very Large Data Bases, September 13-17, 2010, Singapore.
Proceedings of the VLDB Endowment, Vol. 3, No. 1
Copyright 2010 VLDB Endowment 2150-8097/10/09... $ 10.00.

2. PROBLEM DEFINITION

This paper deals with mining interesting phrases in a document collection derived by a query. A phrase is a sequence of terms that appear contiguously in the text. The frequency of a phrase p in a collection of documents $\mathcal{T}$ is denoted by $freq(p, \mathcal{T})$ and defined as the number of documents in $\mathcal{T}$, which contain p. $freq(p, \mathcal{T}) = count\{d : d \in \mathcal{T} \wedge p \in d\}$. Let $\mathcal{D}$ be a document corpus and $\mathcal{D}'$ be an ad-hoc subset of it containing all documents that satisfy a query. A straightforward measure for interestingness of a phrase p with respect to $\mathcal{D}'$ is simply the *local* phrase frequency $freq(p, \mathcal{D}')$. However, this favors common phrases that contain stopwords and have a high frequency in the whole corpus. A more appropriate measure is Definition 1, which normalizes the occurrences of a phrase in $\mathcal{D}'$ by its *global* frequency in $\mathcal{D}$.

DEFINITION 1. *Let $\mathcal{D}'$ be an ad-hoc subcollection of a document corpus $\mathcal{D}$. Let p be a phrase. The interestingness $I_{\mathcal{D}}(p, \mathcal{D}')$ of p w.r.t. $\mathcal{D}'$ is defined by:*

$$I_{\mathcal{D}}(p, \mathcal{D}') = \frac{freq(p, \mathcal{D}')}{freq(p, \mathcal{D})} \quad (1)$$

It may not be realistic to compute $I_{\mathcal{D}}$ for all phrases that appear in any document of $\mathcal{D}'$. So, we define our mining problem as finding the k phrases with the highest interestingness. In the next section, we show how the corpus $\mathcal{D}$ can be pre-processed and indexed, in order to solve this problem.

3. INDEXING AND MINING TECHNIQUES

In this section, we investigate methods for mining interesting phrases. We first outline a baseline method that does not rely on indexing the corpus $\mathcal{D}$. Then, we review the inverted indexing approach of [23] and discuss how it can be applied to solve our problem. Finally, we present in detail our *forward indexing* proposal, investigating alternative ways to order the contents of the lists, which either facilitate compressibility or allow for early termination of top-k interesting-phrase search in $\mathcal{D}'$.

Phrases that occur in very few documents of the corpus $\mathcal{D}$ are expected to give little insight to the analyst. Therefore, we restrict our search to only phrases that exist in a minimum number of documents τ (e.g., τ = 5 or 10). We denote this set of candidate phrases by $\mathcal{C}$. We also limit ourselves to phrases with meaningful length limits, between configurable bounds $minlength$ and $maxlength$. We typically consider $minlength$=2, thus disregarding all individual words – single $terms$ in information-retrieval jargon, but capturing person names and composite nouns. For $maxlength$, 5 would be a canonical choice, as this still considers short catchphrases (e.g., in quotations or slogans) but excludes entire sentences. $\mathcal{C}$ can be computed at a pre-processing phase, by scanning each document $d \in \mathcal{D}$ using a sliding window to extract all phrases of lengths $minlength$ to $maxlength$. Duplicate phrases in d are removed (by sorting or hashing) and then a counter is increased for each phrase in d. Finally, the phrases whose counters are at least τ are inserted to $\mathcal{C}$. As we use a small value of $maxlength$, considering phrases of various lengths simultaneously is more efficient than using a sequence mining approach (e.g., [27]) that would scan the documents multiple times.

3.1 A Baseline Method

A baseline approach for computing the local frequencies in $\mathcal{D}'$ of all phrases in $\mathcal{C}$ is to perform a window-scan over each document $d \in \mathcal{D}'$ and extract all phrases. By keeping a hash map for the phrases, we can count the documents that contain them. In the end, assuming that we have pre-processed and indexed the global frequencies of all phrases (i.e., set $\mathcal{C}$), for each phrase in the hash map, we look up its global frequency and compute $I_{\mathcal{D}}(p, \mathcal{D}')$ according to Definition 1.

This method is expected to be expensive because of the high scanning cost, even for compressed text documents, since it involves tokenization (e.g., identifying sentence boundaries). The cost can be reduced if we replace the original document representations by term-ID arrays, after having defined a mapping between tokens and term IDs at a preprocessing phase on the complete corpus. This way, the token processing cost can be saved. In addition, term IDs can be compressed to save storage. Still, even after this improvement, there is a burden of window-scanning the term-ID arrays and formulating the phrases. If a phrase appears multiple times in the same document, we still need to care for not double-counting it (by hash-set lookups). In addition, document-counters for each phrase have to be maintained and updated during the process. Overall, we need more effective approaches than this baseline method.

3.2 Phrase-Inverted Indexing

As a module of their multidimensional context exploration framework, Simitsis et al. proposed in [23] a method for finding interesting phrases in an ad-hoc subset of a document corpus. Their definition for interestingness (termed *relevance* in the paper) is a normalized version of our Definition 1. Their approach to find the most interesting phrases efficiently is to create a *phrase inverted index* for the corpus as follows. During a preprocessing phase, for each phrase p, an inverted list that contains the IDs of documents that include p is constructed. Now, the local frequency of a phrase p in $\mathcal{D}'$ can be found by intersecting the inverted list of p with $\mathcal{D}'$. The global frequencies are already stored in the heads of the lists (they are equal to the list lengths); therefore, by iterating through *all* inverted lists we can find the interestingness of all phrases and determine the top-k response set. Intersecting $\mathcal{D}'$ with long lists can be accelerated by randomizing the contents of the lists and (i) skipping uniformly over the randomized lists, (ii) stopping after a maximum number of comparisons or when a large enough intersection is already found. This way, an accurate approximation of the local frequency can be obtained (see [23] for more details).

As an example, consider a corpus $\mathcal{D}$ of 20 documents and a set $\mathcal{C} = \{p_1, p_2, \ldots, p_{12}\}$ of phrases having a minimum support $\tau = 4$. Table 1 (left) shows the inverted lists for the phrases in $\mathcal{C}$. The length of each list p_i is shown in brackets and the list contents are the document IDs that contain the phrase. Consider a subset of $\mathcal{D}$ containing documents $\mathcal{D}' = \{d_1, d_4, d_5, d_9, d_{12}, d_{17}, d_{18}, d_{20}\}$. By intersecting $\mathcal{D}'$ with each inverted list we can compute the interestingness of the corresponding phrase, as shown in Table 1 (right). For $k = 2$, the top-k phrases with the highest interestingness are p_2 and p_6, with $I_{\mathcal{D}}(p_2, \mathcal{D}') = 4/4$ and $I_{\mathcal{D}}(p_6, \mathcal{D}') = 5/6$.

The main drawback of this approach is that the whole set of inverted lists must be scanned to obtain the result. In a typical corpus consisting of few thousand documents millions of phrases can be found. In order to avoid scanning all inverted lists, [23] suggest computing the K phrases ($K > k$) with the highest local frequency first, and then post-processing them using the interestingness formula to find the k most interesting phrases in them. To find these K phrases efficiently, they store and access the phrase-inverted lists in *decreasing* global frequency order, and terminate as soon as the K-th smallest local frequency found is greater than the global frequencies of the remaining phrases. However, there is no guarantee that the real k most interesting phrases will be included in the K locally most frequent ones. Consider again the example of Table

Table 1: Example of a phrase-inverted index

phrase	contents	$I_{\mathcal{D}}$
$p_1(4)$	$\{d_2, d_3, d_9, d_{13}\}$	1/4
$p_2(4)$	$\{d_4, d_5, d_{12}, d_{18}\}$	4/4
$p_3(4)$	$\{d_5, d_8, d_{16}, d_{17}\}$	2/4
$p_4(4)$	$\{d_9, d_{12}, d_{16}, d_{19}\}$	2/4
$p_5(5)$	$\{d_4, d_5, d_{12}, d_{16}, d_{19}\}$	3/5
$p_6(6)$	$\{d_3, d_4, d_5, d_9, d_{12}, d_{18}\}$	5/6
$p_7(8)$	$\{d_1, d_2, d_4, d_9, d_{12}, d_{15}, d_{16}, d_{17}\}$	5/8
$p_8(9)$	$\{d_3, d_4, d_8, d_9, d_{12}, d_{14}, d_{17}, d_{18}, d_{20}\}$	6/9
$p_9(10)$	$\{d_1, d_3, d_4, d_5, d_9, d_{10}, d_{12}, d_{17}, d_{18}, d_{19}\}$	7/10
$p_{10}(10)$	$\{d_1, d_2, d_3, d_4, d_5, d_8, d_{10}, d_{12}, d_{17}, d_{20}\}$	6/10
$p_{11}(11)$	$\{d_2, d_4, d_5, d_6, d_9, d_{12}, d_{13}, d_{15}, d_{17}, d_{18}, d_{20}\}$	7/11
$p_{12}(12)$	$\{d_1, d_3, d_4, d_5, d_7, d_9, d_{10}, d_{12}, d_{13}, d_{17}, d_{18}, d_{20}\}$	8/12

1 and assume that we find the $K = 6$ phrases with the largest local frequency first and then the top-k most interesting ones among these for $k = 2$. Finding the K locally most frequent phrases requires scanning all inverted lists of Table 1 in reverse order until p_5. By then, there are already $K = 6$ phrases of local frequency 5 or more (these are p_6 to p_{12}). However, the top-2 interesting phrases p_{12} to p_5 do not include p_2, which is among the two actual most interesting phrases in $\mathcal{D}'$. Therefore, the method of [23] has to be regarded as an approximate method.

3.3 Forward Indexing

Our *forward indexing* approach creates for each document $d \in \mathcal{D}$ a list F_d, which contains the phrases from $\mathcal{C}$ that are included in d. Thus, we replace the (exact) textual content of a document, by the phrases from $\mathcal{C}$ included in it. While in [23] the inverted lists of the vast majority of phrases are intersected with the input set of documents $\mathcal{D}'$, in our approach, we intersect the forward lists only of the documents in $\mathcal{D}'$, in order to obtain the frequency of phrases in $\mathcal{D}'$. The advantage is that the number of accessed lists for finding the frequent phrases directly depends on the size of $\mathcal{D}'$, not $\mathcal{D}$.

The basic algorithm on this scheme (Algorithm 1) performs a $|\mathcal{D}'|$-way merge join, by scanning the sorted input lists in parallel. During the merge, for the current phrase p, the local frequency $freq(p, \mathcal{D}')$ can be counted by the number of inputs where p is seen. The interestingness of p can be determined by accessing its global frequency. Global frequencies can either be explicitly stored in the forward lists (e.g., embedded in phrase IDs) or kept in a separate phrase-dictionary in memory. Thus, we can update the set of top-k interesting phrases after every output of the merge operator and the whole process can terminate with one synchronized scan over the forward lists of the documents in $\mathcal{D}'$. Depending on how the phrases in the forward lists are ordered, in the rest of this section, we investigate different alternatives of our indexing scheme, that improve upon this basic algorithm.

Algorithm 1 Basic Forward Index Search

FWB($\mathcal{D}', k$)
1: $R = \emptyset$ ▷ contains k phrases of highest $I_{\mathcal{D}}(p, \mathcal{D}')$
2: perform $|\mathcal{D}'|$-way merge join of the lists F_d for all $d \in \mathcal{D}'$
3: **for** each output phrase p **do**
compute $freq(p, \mathcal{D}')$, compute $I_{\mathcal{D}}(p, \mathcal{D}')$ and update R to include p if applicable
4: return R

3.4 Frequency-based Ordering and Early Termination

It is possible to adapt Algorithm 1 to avoid scanning the forward lists F_d for each $d \in \mathcal{D}'$ completely, if the order of the phrases p in each list is defined according to their global frequency $freq(p, \mathcal{D})$. That is, less frequent phrases appear first in the forward lists of the documents. While the lists are scanned in parallel by Algorithm 1 (and R is being updated), we know that for any phrase p that has not been seen so far, the maximum possible $I_{\mathcal{D}}(p, \mathcal{D}')$ value is

$$max^p_{I_{\mathcal{D}}} = \min\left\{1, \frac{|\mathcal{D}'|}{freq(p, \mathcal{D})}\right\} \quad (2)$$

Due to the ordering of phrases in the lists, if phrase p precedes phrase q in this order, $max^q_{I_{\mathcal{D}}} \leq max^p_{I_{\mathcal{D}}}$. Therefore, as soon as for the next unexamined phrase p, $max^p_{I_{\mathcal{D}}} \leq \theta$, where θ is the lowest interestingness of phrases in the current top-k set R, the algorithm can safely terminate reporting R as the result. Algorithm 2 shows how Algorithm 1 can be extended along these lines.

Sorting the phrases according to their global frequencies in the forward lists allows for an economic embedding of these frequencies into the lists. Since multiple phrases share the same frequency, we can use a representation that stores the phrases that have the same frequency as a group and the frequency once in the heading of the group. This way, the global frequency needs not be accessed at an external hash table. Table 2 is an illustration of this storage model. Alternatively, global frequencies can be embedded in the identifiers of the phrases, occupying the most significant bits, to force frequency-based ordering and access to frequencies by means of ID decoding.

Algorithm 2 Forward Index Search with Early Termination

FWE($\mathcal{D}', k$)
1: $R = \emptyset$ ▷ contains k phrases of highest $I_{\mathcal{D}}(p, \mathcal{D}')$
2: perform $|\mathcal{D}'|$-way merge join of the lists F_d for each $d \in \mathcal{D}'$
3: **for** each p, output by the merge operator **do**
compute $freq(p, \mathcal{D}')$
compute $I_{\mathcal{D}}(p, \mathcal{D}')$
update R to include p if applicable
$max^p_{I_{\mathcal{D}}} = \min\{1, \frac{|\mathcal{D}'|}{freq(p, \mathcal{D})}\}$
$\theta = k$-th interestingness value in R
if $max^p_{I_{\mathcal{D}}} \leq \theta$ terminate merge-join
4: return R

As an example, let $\mathcal{D}' = \{d_1, d_4, d_5, d_9, d_{12}, d_{17}, d_{18}, d_{20}\}$. Table 2 shows the forward lists of the documents in $\mathcal{D}'$, wherein the phrases are grouped and ordered in ascending global frequency order. Assume that we are looking for the top-2 most interesting phrases. The forward lists are synchronously scanned and merged. At the point after $|\mathcal{D}'|$ equals the global frequency of the last accessed phrase p_{last} from the merger (i.e., $p_{last} = p_7$), we can start applying the early termination test. At this stage, the 2 phrases with the highest interestingness are $R = \{p_2, p_6\}$ and $\theta = freq(p_6, \mathcal{D}') = 5/6$. Since $\theta < |\mathcal{D}'|/freq(p_{last}, \mathcal{D}')$, we cannot terminate, as it is possible to find a phrase later that can enter the top-2 set R. The algorithm terminates after p_9 is found and $|\mathcal{D}'|/freq(p_9, \mathcal{D}') = 8/10 < \theta$. In practice, as demonstrated by our experiments in Section 4, this method can prune a large percentage of the lists, especially if they contain phrases with very high global frequency in their tails.

3.5 Prefix-maximal Phrases and Lexicographic Phrase Ordering

An important issue with the forward indexing scheme is its space requirements. Since all phrases in $\mathcal{C}$ that are contained in a document must be included in the corresponding forward list, the sizes of the lists may grow significantly, even after compressing the phrase representations in them. In this section, we investigate the possibility of reducing the sizes of these lists by avoiding to index all

Table 2: Example of a forward index

list	contents
F_{d_1}	$(9)p_8$ $(10)p_9, p_{10}$ $(12)p_{12}$
F_{d_4}	$(4)p_2$ $(5)p_5$ $(6)p_6$ $(8)p_7$ $(9)p_8$ $(10)p_9, p_{10}$ $(11)p_{11}$ $(12)p_{12}$
F_{d_5}	$(4)p_2, p_3$ $(5)p_5$ $(6)p_6$ $(10)p_9, p_{10}$ $(11)p_{11}$ $(12)p_{12}$
F_{d_9}	$(4)p_1, p_4$ $(6)p_6$ $(8)p_7$ $(9)p_8$ $(10)p_9$ $(11)p_{11}$ $(12)p_{12}$
$F_{d_{12}}$	$(4)p_2, p_4$ $(5)p_5$ $(6)p_6$ $(8)p_7$ $(9)p_8$ $(10)p_9, p_{10}$ $(11)p_{11}$ $(12)p_{12}$
$F_{d_{17}}$	$(4)p_3$ $(8)p_7$ $(9)p_8$ $(10)p_9, p_{10}$ $(11)p_{11}$ $(12)p_{12}$
$F_{d_{18}}$	$(4)p_2$ $(6)p_6$ $(9)p_8$ $(10)p_9$ $(11)p_{11}$ $(12)p_{12}$
$F_{d_{20}}$	$(9)p_8$ $(10)p_{10}$ $(11)p_{11}$ $(12)p_{12}$

phrases in them.

The main idea comes from the monotonicity of $freq(p, \mathcal{D})$ with respect to the set-containment relationships between phrases. If a phrase p is included in a document d, all sub-phrases $p' \subset p$ should also be present in d. Therefore, if we include p in the forward list F_d of d, it is not necessary to add any $p' \subset p$ there; the contents of the forward lists can be minimized, if we include only the maximal-length phrases (in $\mathcal{C}$) that are present in each document.

Figure 1(a) shows the contents of three documents. For ease of presentation, we denote distinct terms by characters and the content of a document as a string (i.e., sequence of terms). Figure 1(b) shows the set of distinct phrases of length $minlength = 1$ to $maxlength = 4$ that occur in each document. Assuming that $\mathcal{C}$ contains only phrases up to length 4 and that all such phrases are included in $\mathcal{C}$, Figure 1(c) shows the maximal-length phrases in each document. Clearly, if we include only these, the lengths of the forward lists can be greatly reduced.

On the other hand, this compressed representation complicates evaluation. Now, computing $freq(p, \mathcal{D}')$ for an arbitrary phrase p cannot be achieved by simply scanning (and intersecting) the forward lists (i.e., using Algorithm 1). For example, assuming that $\mathcal{D}' = \{d_1, d_2, d_3\}$, it is not clear how the local frequency of phrase "bef" can be obtained by intersecting the lists shown in Figure 1(c). Fortunately, there is a forward-list format that avoids indexing the majority of non-maximal phrases and allows computing the frequencies of all phrases efficiently at a single scan of the lists. Our proposal is based on the concept of *prefix-maximal* phrases which we define below:

DEFINITION 2. *A phrase p is prefix-maximal w.r.t. document $d \in \mathcal{D}$, if (i) $p \in \mathcal{C}$ (i.e., p is globally frequent), (ii) $p \in d$, and (iii) there exists no phrase $p' \in \mathcal{C}$, such that p is a prefix of p' and $p' \in d$.*

For example, in Figure 1(a), phrase "abc" is not prefix-maximal w.r.t. d_1 because d_1 also contains "abcd" and "abc" is a prefix of "abcd".

Now, we adapt the forward index to contain for each document d only the prefix-maximal phrases w.r.t. d. In addition, in each forward list, the phrases are ordered lexicographically. Figure 1(d) shows the prefix-maximal forward lists for the documents of Figure 1(a). Observe that the lists are only slightly larger than those containing only maximal phrases (Figure 1(c)), but significantly smaller than the lists containing all phrases (Figure 1(b)).

Having constructed this index, the challenge is to adapt Algorithm 1 to compute the interestingness of all phrases contained in $\mathcal{D}'$. This is not straightforward, as we need to avoid over-counting of subphrases that are not explicitly stored in the forward lists but implied by prefix-maximal phrases which contain them. Algorithm 3 is a pseudocode of this adaptation. The lists are merged, and the prefix-maximal phrases at all inputs are retrieved in lexicographical order. Figure 1(e) shows the order by which the phrases are retrieved when merging the lists of Figure 1(d) — ignore the numbers in parentheses for the moment. At each access, the algorithm identifies the changes in the prefix of the current phrase p, compared to the previous phrase p_{prev} (Line 10).[1] Specifically, it finds the longest common prefix s between p and p_{prev}. Every prefix of p_{prev} that is longer than s corresponds to a phrase that can never be seen subsequently by the algorithm (because prefix-maximal phrases are sorted lexicographically in the lists). Thus, the local frequencies of all these prefixes in $\mathcal{D}'$ can be immediately determined and output (Lines 13–16).

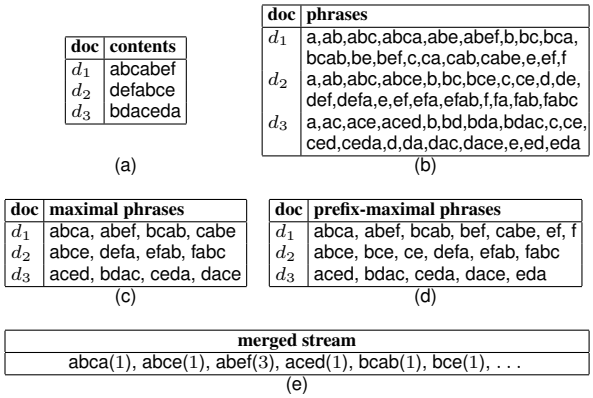

Figure 1: Example of prefix-maximal lists

In order to compute the local frequencies of all prefixes correctly, we keep a list of counters C, one for each possible phrase length. For each new prefix-maximal phrase p encountered, say from document d_j we check its difference to the previous phrase p^j_{prev} seen at document d_j (Line 9). For all positions in s, where p_{prev} and p^j_{prev} differ, we increase the corresponding counter, in order to take into consideration that the corresponding prefixes have been seen at d_j (Line 11–12) and avoiding over-counting for the prefixes where p_{prev} and p^j_{prev} are common (these have been counted when p^j_{prev} was considered). When comparing p with p_{prev}, for all prefixes that are different, we compute the local frequencies of the corresponding prefixes using the counters (Lines 13–16). Then, for the current phrase p, we initialize the counters for the new prefixes to 1. Finally, after the last phrase is accessed from the merged input, we use the existing counters to count and output the frequencies of all its prefixes (Lines 20–23). Summing up, Algorithm 3 correctly computes the local frequencies of all phrases by merging the forward lists containing the prefix-maximal phrases.

p^j_{prev} can be computed if the algorithm (or the merger) keeps track of the previous phrase seen at each input. In our implementation, we avoid its computation by embedding this information in the inverted lists. That is, together with the phrase representation, we also store for each phrase the first position where it differs from the previous phrase in the same list. This information is also used for compressing the lists, as it indicates the common prefix between consecutive phrases; this prefix needs not be repeated in the representation of the next phrase. Figure 1(e) shows, for each phrase p that is produced by the merger, the first position where p differs with p^j_{prev} in brackets.

As an example, let us see how the local frequencies of all phrases in $\mathcal{D}' = d_1, d_2, d_3$ can be computed, using the forward index of Figure 1(d). First, we initialize an array of 4 counters, assuming that the minimum (maximum) length of a phrase in $\mathcal{C}$ is 1 (4). The first phrase that comes from the merger (see Figure 1(e)) is "abca" from input d_1, therefore we set p_{prev} ="abca" and add 1 to all 4

[1]Since we merge multiple inputs, p could be identical to p_{prev}.

Algorithm 3 Merging Prefix-Maximal Forward Lists

```
    FWP(D', k)
 1: C = new array of counters                          ▷ initially 0
 2: p = first phrase from merger
 3: j = input where p was seen
 4: for all positions i of p do
 5:     C[i] = C[i] + 1          ▷ count prefixes of p seen at j-th input
 6: p_prev = p                                        ▷ track prev. phrase
 7: while p = next phrase from merger do              ▷ while more phrases
 8:     j = input where p was seen
 9:     jpos = first position where p and p^j_prev differ
10:     dpos = first position where p and p_prev differ
11:     for i = jpos to dpos-1 do                    ▷ count current input
12:         C[i] = C[i] + 1
13:     outp = first dpos − 1 terms of p_prev        ▷ common prefix
14:     for i = dpos to len(p_prev) do               ▷ for each diff. prefix
15:         append p_prev[i] to outp
16:         freq(outp, D') = C[i]                    ▷ set frequency
17:     for i = dpos to len(p) do                    ▷ reset counters
18:         C[i] = 1
19:     p_prev = p                                   ▷ track prev. phrase
20: outp = ∅                          ▷ handle all prefixes of last phrase
21: for i = 1 to len(p_prev) do
22:     append p_prev[i] to outp
23:     freq(outp, D') = C[i]
```

counters. The next phrase is "abce" (seen at input $j = d_2$) and the first positions where it differs with p^j_{prev} and p_{prev} are $jpos$=1 and $dpos$=4, respectively (p^j_{prev} is null, as p is the first phrase seen at input $j = d_2$). The algorithm will increase the counters for positions 1 though 3, reflecting that the prefixes "a", "ab", and "abc" have been seen also at input $j = d_2$. In addition, "abca" (i.e., the common prefix "abc" of p_{prev} and p padded with the last term of p_{prev}) will be output, with frequency $C[4]$=1. Next, counter $C[4]$ is reset to 1 and p_{prev} are updated to "abce", before accessing the next phrase p ="abef" from the merger. This phrase differs at positions $jpos$=3 and $dpos$=3 from p^j_{prev}="abca" and p_{prev}="abce", respectively. No counters are updated (since $jpos$=$dpos$), and the algorithm outputs "abc" and "abce" with frequencies 2 and 1, respectively, while resetting $C[3]$ and $C[4]$ to 1. Note that at any stage the counters will correctly capture the number of inputs where all prefixes of p_{prev} have been seen.

Algorithm 3 computes only the local frequencies of phrases (at Lines 16 and 23) but not their interestingness. The global frequencies of the non-maximal phrases (required for the computation of interestingness) are not explicitly included in the index, so for the interestingness of each phrase to be computed we need to do a look-up at a hash table that stores the global frequencies of all phrases. This table typically fits in memory and searches are efficient.

Summing up, Algorithm 3 operates on a very economical representation of the forward lists. On the other hand, it has to exhaustively access all merged lists, being unable to apply early termination heuristics like Algorithm 2, due to the lexicographic ordering of phrases in the lists as required for this method.

4. EXPERIMENTAL EVALUATION

We used the recently released annotated New York Times corpus [21] for our experimental evaluation, which includes more than 1.8 million New York Times articles published between 1987 and 2007. The raw size of the textual content is about 8 GB. The dataset includes rich metadata (e.g., an article's publication date, topical classification, and author biography), as well as, annotations (e.g., persons, organizations, and places mentioned in an article).

All methods described in this paper were implemented in Java (using SUN JDK 1.6). Experiments were run on a SUN server-class machine having 16 GB of main memory, four AMD Opteron single-core CPUs, a large network-attached RAID-5 disk array, and running Windows Server 2003.

For processing keyword queries, we employ conjunctive semantics, i.e., all query keywords are mandatory for a document to be reported as a result. Disjunctive queries were also tested, but we do not report the results here, as no significant performance difference was observed for them. We use Okapi BM25 [20, 22] (using the established parameter setting $k_1 = 1.2$ and $b = 0.75$), as a state-of-the-art relevance model to rank result documents. Further, we index the dataset such that metadata and annotations provided with it can easily be leveraged at query time. For example, in the terms of a document with publication date March 11th we add the artificial term publication_date$March_11th,_1999. Notice, though, that these artificial terms are hidden from the relevance model and thus do not affect the relative ranking of results.

Table 3 shows the different indexing methods and their representations that we compare against each other. *TermID* is a baseline method which replaces each document by a sequence of term IDs. *IntArray* models phrases as term ID arrays (e.g., "Steve Jobs" is translated to [441,312], assuming that *Steve* is term 441 and *Jobs* is term 312). Phrases in forward lists are ordered lexicographically and Algorithm 1 is used for merging. *FreqIntArray* encodes phrases as term ID arrays, phrases in lists are sorted in decreasing global frequency order, and the global frequency of each phrase is kept explicitly in the list, as explained in Section 3.4 and illustrated in Table 2. Algorithm 1 is used. In *PhraseID*, we globally assign phrase IDs, according to their global frequency. For each phrase p, we set the first 32 bit-block of its phrase ID (of type long) to the global frequency $freq(p, \mathcal{D})$. The second 32 bit-block is used to break ties among phrases having the same global frequency. *EarlyTerminationFreqIntArray* and *EarlyTerminationPhraseID*, use the same representation as FreqIntArray and PhraseID, respectively, but Algorithm 2 is used instead of Algorithm 1. *PrefixMaxIntArray* is similar to IntArray, but only prefix-maximal phrases are stored in the lists and Algorithm 3 is used. We compare our methods to an implementation of *MCX*, in accordance to [23], where we first find the K=1000 phrases with the highest local frequency using the index and then post-process them to find the k most interesting ones. The maximum number of comparisons per approximate list intersection is set to $M = 64$.

4.1 Queries

We defined a set of benchmark queries, which we consider to reflect the envisioned application. Our queries, as shown in Figure 2 with their respective result sizes, are subdivided into three categories, namely, (i) person-related queries, (ii) news-related queries, and (iii) location-related queries. The first two categories are standard keyword queries, whose choice was inspired by Google's Zeitgeist [10] service. For the third category, we make use of the topical classification provided with the dataset.

4.2 Index Sizes

Our first experiment compares the different methods with regard to the size of their corresponding indexes. For each of the methods, we generated the corresponding indexes for different values of the minimum support threshold (τ); in each case, we only index phrases whose global frequency in the corpus $\mathcal{D}$ is at least τ. In all cases, we index only phrases of lengths between 2 and 5.

As Figure 3 shows, the TermID representation is the most economical (consuming about 1.8 GB of space), as only the terms are

(i) Person-Related Keyword Queries
jennifer lopez (1,130), osama bin laden (5,951), eminem (796), steve jobs (2,982), kobe bryant (1,385), paris hilton (1,119), harry potter (2,053), martha stewart (3,044), camilla parker bowles (173), prince charles (4,019), barack obama (867), hillary clinton (11,073), rudy giuliani (2,485), martin luther king (6,533), george harrison (2,494)

(ii) News-Related Keyword Queries
world trade center (25,562), american airlines (14,187), world cup (17,974), winter olympics (4,357), korea (22,474), hurricane katrina (3,515), american idol (3,009), tsunami (1,452), bankruptcy (21,188), iphone (82), presidential election (26,492), aol time warner merger (588), sars (1,153), space shuttle columbia (1,322), tour de france (3,328)

(iii) Location-Related Metadata Queries
/travel/guides/destinations/europe (94,789)
/travel/guides/destinations/middle_east (58,457)
/travel/guides/destinations/north_america (325,300)
/travel/guides/destinations/ asia (62,290)
/travel/guides/destinations/africa (22,592)
/travel/guides/destinations/europe/france (12,874)
/travel/guides/destinations/europe/france/rhone_valley/lyon (40)
/travel/guides/destinations/europe/germany (11,165)
/travel/guides/destinations/europe/italy (6,722)
/travel/guides/destinations/central_and_south_america (20,466)
/news/world/africa (14,961)
/news/world/asia_pacific (43,886)
/news/world/americas (26,710)
/news/world/europe (54,230)
/news/world/middle_east (46,020)

Figure 2: Keyword & metadata queries (number of query results given in parentheses)

Table 3: Query methods evaluated

method	description
TermID	baseline method, Section 3.1
IntArray FreqIntArray PhraseID	Section 3.3
EarlyTerminationFreqIntArray EarlyTerminationPhraseID	Section 3.4
PrefixMaxIntArray	Section 3.5
MCX	[23] (reviewed in Section 3.2)

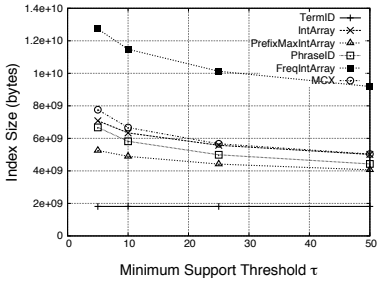

Figure 3: Index sizes (bytes) for different values of τ

included in each list. The PrefixMaxIntArray and PhraseID representations follow. Even though PrefixMaxIntArray indexes only prefix-maximal phrases, which are roughly equal to the number of terms, because these are represented as integer arrays that occupy a few bytes, the representation takes more space than TermID. PhraseID indexes all phrases, but keeping only a phrase identifier for each of them, so its overhead over PrefixMaxIntArray is not high. The phrase-inverted index employed by MCX and the IntArray representation yield indexes that consume about the same amount of space. Finally, the FreqIntArray representation produces indexes that are considerably larger than the other methods. This is not surprising, given that, in comparison to IntArray, the method keeps additional information and suffers from lower compressibility, as the phrases in a list are grouped primarily by global frequency and there are large gaps between consecutive phrases in the same group.

4.3 Performance Comparison

In this set of experiments we examine the different methods with regard to their query-processing performance. First, we investigate how the cardinality of $\mathcal{D}'$ affects performance. We vary the cardinality of the input $\mathcal{D}'$ between 100 and 10,000. We do this by using only queries with larger results and truncating them to their highest ranked 100 to 10,000 documents according to the per-query relevance ranking. We fix the result size for the interesting phrases as $k = 100$ and the minimum support threshold as $\tau = 10$.

Figures 4 reports the mean wallclock time per query (measured with warm caches) for the different approaches. MCX is not affected by the input cardinality $|\mathcal{D}'|$. It requires about 100 seconds for a single query. In contrast, all other methods benefit from smaller input cardinalities. For $|\mathcal{D}'| = 100$ our best method EarlyTerminationPhraseID outperforms MCX by three orders of magnitude. For $|\mathcal{D}'|$ =1,000 the improvement over MCX is still more than a factor of 128. For larger $|\mathcal{D}'|$ MCX becomes competitive to some of our methods like IntArray, but is still outperformed by EarlyTerminationFreqIntArray by a factor of 10. These results show that MCX will only be competitive for very large input cardinalities, but for such queries the phrase analysis cannot be done in real-time, so its applicability is limited. On the other hand, for our test corpus and the practical, medium-sized query results of our benchmark, the forward-index-based methods are always the method of choice. A full-fledged system should perhaps implement both methods and let a query optimizer decide which method to use.

Figure 7 shows the amount of data read at query time for the same experiment. We observe a linear increase for all forward indexing methods starting at 200 KB for $|\mathcal{D}'| = 100$ up to 6 MB for $|\mathcal{D}'|$ =10,000. In contrast, MCX needs to read at least 3 GB in all cases. For MCX we may also observe a slight decrease when increasing $|\mathcal{D}'|$. This is due to the fact that for larger $|\mathcal{D}'|$ this method may stop earlier, as the top-K locally frequent phrases can be found faster.

In a second experiment, we fix the result size as $|\mathcal{D}'| = 500$, by taking only the first 500 results of queries that exceed this selectivity, and vary the minimum support threshold τ. In all cases, we determine the $k = 100$ most interesting phrases. Figures 5 and 8 report the mean wallclock time per query (measured with warm caches) and the mean amount of data read per query for the different approaches, respectively. We skipped MCX from this comparison because its cost exceeded 100 seconds at all times.

The results in Figure 5 show that the baseline method TermID requires about 4 seconds to compute a query. In contrast, the different forward indexing methods require at most 1.3 seconds for all parameters. Among our methods, EarlyTerminationPhraseID performs best requiring only 250 ms. This is by a factor of 8 better than the baseline method. In addition, this method also outperforms the other forward indexes like PrefixMaxIntArray which takes about 1,000 to 750 ms. Although a worst-case theoretical analysis (see Appendix A) suggests that PrefixMaxIntArray would perform better, EarlyTerminationPhraseID is the winner because of (i) the ultra-fast merging of phrase-IDs, (ii) its higher compressibil-

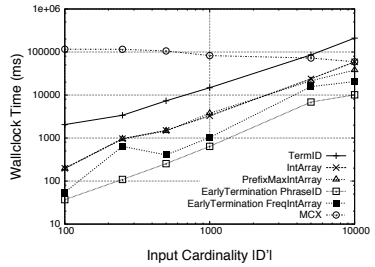

Figure 4: Wallclock times (ms) for different values of $|\mathcal{D}'|$

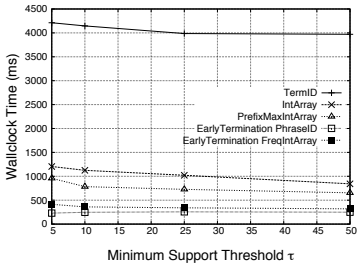

Figure 5: Wallclock times (ms) for different values of τ

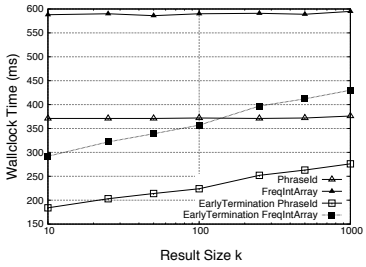

Figure 6: Wallclock times (ms) for different values of k

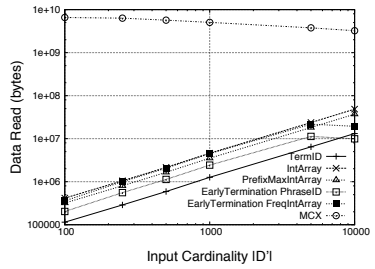

Figure 7: Bytes read for different values of $|\mathcal{D}'|$

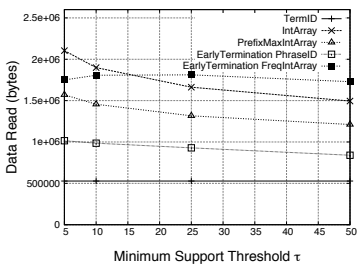

Figure 8: Number of bytes read for different values of τ

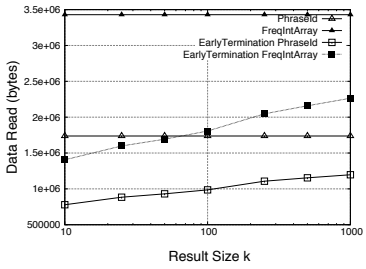

Figure 9: Number of bytes read for different values of k

ity, and (iii) the early termination heuristic.

Figure 8 shows the amount of data read during query processing for the same experiment. Although the baseline TermID method reads the fewest bytes (not surprisingly, as it only has to read the termID representations of the documents), it has much higher query cost than the forward indexing methods, as discussed above. From these methods, EarlyTerminationPhraseID performs best, due to its high compressibility when compared to integer-array phrases and the effect of early termination. For EarlyTerminationFreqIntArray we see a slight increase from $\tau = 5$ to $\tau = 25$. For very low values of τ phrases might qualify as interesting if their global frequency is very low. Thus their interestingness score is very high, which is favorable for the early termination. Since these globally infrequent phrases tend to be long, EarlyTermination achieves higher savings in terms of the amount of data read, when applied on the FreqIntArray representation that keeps phrases explicitly as integer arrays. With increasing τ these globally infrequent phrases are ruled out from both representations.

In general, the influence of parameter τ is relatively mild. This is because after the initial pruning of extremely rare phrases (the long tail of phrases that occur only once or twice), removing more infrequent phrases does not affect the lengths of the forward-index lists too much. Because of these phrases being infrequent, their probability of occurring in the query-result documents is much lower than the occurrence probability of a frequent phrase.

4.4 Effectiveness of Early Termination

Our last experiment studies the effectiveness of the early-termination method introduced in Section 3.4. For this, we fix the minimum support threshold as $\tau = 10$ and the cardinality of the input as $|\mathcal{D}'| = 500$ and vary the number k of interesting phrases to be determined. Figures 6 and and 9 report the mean wallclock times and mean amount of data read by the FreqIntArray and PhraseID representations, with and without early termination. For $k = 10$, EarlyTerminationPhraseID reads about 0.7 MB in 184 ms, while for $k = 500$ it reads 1.1 MB in 263 ms. This is an increase of about 57% and 42%, respectively, for a 50-fold increase of k, quite a remarkably good result. Comparing the early-termination methods with their merge-based counterparts, one can observe significant reductions both in terms of data read and wallclock time even for large values of k.

5. RELATED WORK

Text analytics is increasingly gaining attention due to its value in gathering business intelligence. Previous efforts in this regard have primarily focused on term-level analytics, identifying terms that are specific to a group of documents [6, 16, 24]. Some [7, 8, 13, 15, 19] have taken a multi-dimensional view of the text collection and proposed OLAP-style models for performing drill-down/roll-up on text databases, but remain at the level of terms.

While term-level analytics is important, it is not sufficient for identifying important entity names or marketing slogans which have tremendous value for business intelligence. Early approaches [1, 17] fall short in scaling to large-scale text collections or consider only the collection as a whole but no ad-hoc document sets. Only recently, the MCX system [23] was proposed as an important step in the direction of scalable phrase-level analytics. As we explained earlier, their framework identifies only the most frequent phrases in a given set of documents, and then re-ranks these phrases based on their interestingness. Further, as our results clearly show their choice of using inverted indexes does not scale very well.

The idea of extracting phrases dynamically has also been explored by other communities. In [25] the use of phrases as a means to guide the user through search results is studied. Since their fo-

cus is on at most the top-30 most relevant documents, scalability issues, as we address them, do not play a role in their application. In [26] point-wise mutual information (PMI), as a measure of a phrase's statistical surprisingness, is used to identify key phrases in a document. Thus identified phrases from a document are used as a signature of the document. The signature is then used for querying the collection for similar or related documents. Our framework can be easily adapted for the PMI-based interestingness measure.

Faceted search [12], as pioneered in the Flamenco project [9], is akin to our work in dynamically aggregating information for ad-hoc sets of documents. Although recent extensions [3, 5] move beyond mere count-based aggregation of facets, the key difference to our approach remains that facets are pieces of meta data about the documents. Our approach, in contrast, operates directly on the document contents.

In addition to scalable phrase-level analytics, good visualizations are important for comprehending the results in an intuitive manner. MemeTracker [18], TagLines [6], and BlogScope [2] have explored this for visualizing bursty phrases, terms, or social annotations over time. Similar interfaces could easily be built on top of our framework, and would then enable exploration of interesting phrases for ad-hoc document sets in a scalable and interactive manner, which is not possible in any of the aforementioned systems.

Finding interesting phrases is related to mining frequent subsequences in sequence collections. However, the objectives of the two problems are different, as we count the phrase (i.e., subsequence) frequencies in an *ad-hoc subset* $\mathcal{D}'$ of the corpus $\mathcal{D}$, by preprocessing and indexing the *whole* corpus.

The vertical sequence mining approach [27] uses the same idea of inverted indexing as MCX [23], but a inverted list is created for each element (i.e., term) as opposed to subsequence (i.e., phrase). Data structures similar to forward indexing in data mining, like the FP-tree [11], have different objectives (i.e., compression of multiple transactions for faster itemset support computation vs. early termination in our setting). Accessing such a data structure in a branch-and-bound manner in order to find top-k discriminative patterns has been proposed in [4]. Previous work in data mining which is the most related to ours is [14]. Given two collections $\mathcal{D}_{pos}$ and $\mathcal{D}_{neg}$ of sequences the objective is to find *distinguishing* subsequences that have high frequency in $\mathcal{D}_{pos}$ and low frequency in $\mathcal{D}_{neg}$. The method of [14] could be used to solve our interesting phrase mining problem, by setting $\mathcal{D}_{pos} = \mathcal{D}'$, using $\mathcal{D} \setminus \mathcal{D}'$ as $\mathcal{D}_{neg}$, and keeping track of the top-k results while counting the supports of the phrases. However, this approach would be very expensive. The sequences are generated and counted one-by-one, and counting the frequency of a sequence requires scanning the positions of its previous-level subsequences in all documents of $\mathcal{D}'$.

6. CONCLUSION

This paper has presented scalable techniques for interesting phrase mining from large text corpora. In contrast to the state-of-the-art method MCX [23], which solves the problem only approximately and is inefficient for large corpora, our forward indexing method scales very well with the corpus size. This is the decisive property to enable users to perform interesting phrase analyses on large real world datasets. We have provided several variants of forward indexing and discussed the different trade-offs w.r.t. index size and query response times. Our performance study used a large-scale real-world corpus consisting of 1.8 million articles from New York Times. We compared the different variants of our method against MCX. The results confirm that our forward indexing scheme outperforms MCX by orders of magnitude when the cardinality of the documents to be analyzed is not impractically large.

As discussed in Appendix B, our methods are also applicable to alternative definitions of phrase interestingness. In terms of future work, we are planning to explore other more high-level query types enabled by our method including: timeline interesting phrase analysis, phrase groupings, as well as, an interesting phrase CUBE operator.

7. REFERENCES

[1] H. Ahonen. Knowledge Discovery in Documents by Extracting Frequent Word Sequences. *Library Trends*, 48(1), 1999.

[2] N. Bansal and N. Koudas. BlogScope: A System for Online Analysis of High Volume Text Streams. In *VLDB*, 2007.

[3] O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev. Beyond Basic Faceted Search. In *WSDM '08*, 2008.

[4] H. Cheng, X. Yan, J. Han, and P. S. Yu. Direct discriminative pattern mining for effective classification. In *ICDE*, pages 169–178, 2008.

[5] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman. Dynamic Faceted Search for Discovery-driven Analysis. In *CIKM*, 2008.

[6] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. Visualizing Tags over Time. *ACM Trans. Web*, 1(2):7, 2007.

[7] R. Fagin, R. Guha, R. Kumar, J. Novak, D. Sivakumar, and A. Tomkins. Multi-Structural Databases. In *PODS*, 2005.

[8] R. Fagin, P. Kolaitis, R. Kumar, J. Novak, D. Sivakumar, and A. Tomkins. Efficient Implementation of Large-scale Multi-structural Databases. In *VLDB*, 2005.

[9] Flamenco Project `http://flamenco.berkeley.edu`.

[10] Google Zeitgeist `http://www.google.com/press/zeitgeist.html`.

[11] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD Conference*, pages 1–12, 2000.

[12] M. A. Hearst. Clustering versus Faceted Categories for Information Exploration. *Commun. ACM*, 49(4):59–61, 2006.

[13] A. Inokuchi and K. Takeda. A Method for Online Analytical Processing of Text Data. In *CIKM*, 2007.

[14] X. Ji, J. Bailey, and G. Dong. Mining minimal distinguishing subsequence patterns with gap constraints. In *ICDM*, 2005.

[15] S. Keith, O. Kaser, and D. Lemire. Analyzing Large Collections of Electronic Text Using OLAP. *CoRR*, abs/cs/0605127, 2006.

[16] J. Kleinberg. Bursty and Hierarchical Structure in Streams. In *KDD*, 2002.

[17] B. Lent, R. Agrawal, and R. Srikant. Discovering Trends in Text Databases. In *KDD*, 1997.

[18] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *KDD*, 2009.

[19] C. X. Lin, B. Ding, J. Han, F. Zhu, and B. Zhao. Text Cube: Computing IR Measures for Multidimensional Text Database Analysis. In *ICDM*, 2008.

[20] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[21] New York Times Annotated Corpus `http://corpus.nytimes.com`.

[22] S. E. Robertson and S. Walker. Okapi/Keenbow at TREC-8. In *Text Retrieval Conference*, 1999.

[23] A. Simitsis, A. Baid, Y. Sismanis, and B. Reinwald. Multidimensional Content eXploration. *PVLDB*, 1(1):660–671, 2008.

[24] Y. Sismanis, B. Reinwald, and H. Pirahesh. Document-Centric OLAP in the Schema-Chaos World. In *BIRTE*, 2006.

[25] R. W. White, J. M. Jose, and I. Ruthven. Using Top-Ranking Sentences to Facilitate Effective Information Access. *JASIST*, 56(10):1113–1125, 2005.

[26] Y. Yang, N. Bansal, W. Dakka, P. Ipeirotis, N. Koudas, and D. Papadias. Query by Document. In *WSDM*, 2009.

[27] M. J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning*, 42(1/2):31–60, 2001.

[28] J. Zobel and A. Moffat. Inverted Files for Text Search Engines. *ACM Comput. Surv.*, 38(2):6, 2006.

APPENDIX

A. SPACE AND TIME COMPLEXITIES OF THE ALGORITHMS

All algorithms, except for the baseline method (Section 3.1), have low memory requirements as no bookkeeping for temporary counting of phrase frequencies is required by them. The baseline method scans the documents in $\mathcal{D}'$ one-by-one and generates all phrases dynamically, while counting their frequencies in a hash table. The time complexity of this method is high, as $O(\rho \cdot T(\mathcal{D}'))$ phrases are generated and counted. The methods that operate on the phrase-inverted index (Section 3.2) must scan the whole index in the worst case and perform intersections of $\mathcal{D}'$ with all lists. This results in a very high $O(|\mathcal{C}| \cdot |\mathcal{D}'| + \rho \cdot T(\mathcal{D}))$ worst-case cost, as $\mathcal{D}'$ must be read and intersected $|\mathcal{C}|$ times and all postings in the index should be read. In practice, the cost is lower, due to approximate counting and early termination possibilities.

The worst-case cost of Algorithm 2 (and the simpler Algorithm 1) is that of accessing all $|\mathcal{D}'|$ forward lists, of average length λ. During the merging of the lists, and each time a posting is produced by the merger, the merger itself requires $\log |\mathcal{D}'|$ time to update its priority queue. Thus, the overall cost of Algorithms 1 and 2 is $O(|\mathcal{D}'| \log |\mathcal{D}'| \cdot \lambda)$. For Algorithm 3, the cost at each iteration is dominated by looping over the positions of the current phrase p and comparing it with p_{prev} to determine $dpos$. Two more loops are required to generate the prefixes to be output and reset the counters. Thus, the cost per access from the merger is $O(\rho)$. Assuming that the average length of a forward list in the prefix-maximal index of Section 3.5 is $\lambda' < \lambda$, there are $|\mathcal{D}'|$ lists and $|\mathcal{D}'|\lambda'$ accesses in total. Thus, the overall cost is $O(|\mathcal{D}'| \cdot \lambda'(\rho + \log |\mathcal{D}'|))$. Typically $\rho < \log |\mathcal{D}'|$, thus theoretically Algorithm 3 is more efficient than Algorithm 2, while also requiring less space.

B. OTHER DEFINITIONS OF INTERESTINGNESS

Here, we discuss alternatives to Definition 1 for assessing the interestingness of phrases and explain how our forward indexing approaches can be used for each of these measures.

PMI-based surprising frequency. Point-wise mutual information (PMI) is a measure from information theory for quantifying surprise in the co-occurrence of terms, based on their joint (multivariate) probability versus independent (univariate) probabilities [26]. Based on this, Definition 3 measures interestingness of a phrase by dividing its frequency in $\mathcal{D}'$ by the product of the individual terms' frequencies in $\mathcal{D}$.[2]

DEFINITION 3. *Let $\mathcal{D}'$ be an ad-hoc subcollection of a document corpus $\mathcal{D}$. Let p be a phrase. The interestingness $I_{\mathcal{D}}(p, \mathcal{D}')$ of p w.r.t. $\mathcal{D}'$ is defined by:*

$$I_{\mathcal{D}}(p, \mathcal{D}') = log \frac{freq(p, \mathcal{D}')}{\Pi_{t \in p} prob(t, \mathcal{D})} \quad (3)$$

where $prob(t, \mathcal{D})$ denotes the probability of term t to occur in a random document from $\mathcal{D}$.

We can compute the interestingness of p without having to know its global frequency $freq(p, \mathcal{D})$, but only the global frequencies of the terms, which are much fewer than the phrases that appear in the corpus. All methods discussed in Section 3 can be implemented to apply Definition 3, if we replace the global frequencies of the phrases (wherever they are stored in the indexes) by the corresponding normalized term probability products. These products can either be pre-processed and incorporated in the index or computed on-the-fly by look-ups against a term dictionary.

Differential frequency to another query result. Instead of examining interestingness with respect to the whole corpus $\mathcal{D}$, an analyst may wish to identify phrases that occur surprisingly frequently in $\mathcal{D}'$ compared to another ad-hoc subset $\mathcal{D}''$ of $\mathcal{D}$, which is derived by another query. $\mathcal{D}'$ and $\mathcal{D}''$ could have any set-relationship (e.g., disjoint, overlap, containment). For instance, assume that we are interested in phrases that appear surprisingly frequently in documents $\mathcal{D}'$ relevant to "Steve Jobs" compared to documents $\mathcal{D}''$ relevant to "Bill Gates". For this purpose we can use the following definition:

DEFINITION 4. *Let $\mathcal{D}'$ and $\mathcal{D}''$ be two ad-hoc subcollections of a document corpus $\mathcal{D}$. Let p be a phrase. The interestingness $I_{\mathcal{D}''}(p, \mathcal{D}')$ of p w.r.t. $\mathcal{D}''$ is defined by:*

$$I_{\mathcal{D}''}(p, \mathcal{D}') = \frac{1 + freq(p, \mathcal{D}')}{1 + freq(p, \mathcal{D}'')} \quad (4)$$

We add 1 to both frequencies in order to prevent division by zero if $freq(p, \mathcal{D}'')$ is 0. Finding the most interesting phrases based on this definition using the suggested techniques is still possible, however, the early termination heuristics may not be applicable. The reason is that the phrases in the forward lists cannot be preprocessed and ordered with respect to any ad-hoc subset $\mathcal{D}''$ of the corpus. Therefore, we need to first find the local frequencies of *all* phrases in $\mathcal{D}'$ and $\mathcal{D}''$, intersect them, and pick the top-k interesting phrases according to Definition 4. Note that all methods described in Section 3 can be used to compute the local frequencies of all phrases, by disregarding global frequencies and any pruning/termination condition. More importantly, our forward indexing approaches facilitate evaluation in an operator-based fashion. If the same algorithm is applied twice as two operators that compute the local phrase frequencies in $\mathcal{D}'$ and $\mathcal{D}''$ in parallel, the results from the two operators can be pipelined to a merger, which computes the interestingness of each phrase on-the-fly (as phrases are examined in the same order) and updates the top-k set.

OLAP-style frequency analysis. Finally, our measures (e.g., Definition 1) can be applied at different partitions of a document set, facilitating an OLAP-style analysis of interestingness, as shown below.

DEFINITION 5. *Let $\mathcal{D}'$ be a subcollection of a document corpus $\mathcal{D}$ comprising the documents that satisfy a keyword query q. Let $\mathcal{G}_i, i = 1, \ldots, m$ be a partitioning of $\mathcal{D}'$ into m groups, according to a combination of dimensional axes (e.g., time). Let p be a phrase. We can define the interestingness $I_{\mathcal{D}'}(p, \mathcal{G}_i)$ of p w.r.t. group $\mathcal{G}_i$, as follows:*

$$I_{\mathcal{D}'}(p, \mathcal{G}_i) = \frac{freq(p, \mathcal{G}_i)}{freq(p, \mathcal{D}')} \quad (5)$$

Similarly to Definition 4, we can directly apply the forward list merging approaches to find the most interesting phrases in each group. Again, early termination is not possible, since the denominator refers to an ad-hoc document subset that is not known at index time. While the forward lists of all documents in $\mathcal{D}'$ are merged, we compute for each phrase one *group* counter per $\mathcal{G}_i$, plus a local frequency counter for the whole $\mathcal{D}'$. For each phrase that is encountered, we can immediately compute its interestingness by applying

[2] In the classic PMI definition, the frequencies in the numerator and denominator refer to the same set (i.e., either $\mathcal{D}$ or $\mathcal{D}'$). Definition 3 complies with our interestingness concept, where the frequency of a phrase in $\mathcal{D}'$ is divided by the expected frequencies of its terms in the whole corpus.

Definition 5 for each group. At the same time, we maintain for each group the set of k most interesting phrases seen so far.

Weighted frequencies. All definitions above assume that the documents in $\mathcal{D}'$ (or $\mathcal{D}''$) have the same influence in phrase interestingness. Nonetheless one could argue that if these documents are obtained together with a relevance score (e.g., from a keyword query), then we should weigh their contribution to the interestingness of phrases, accordingly. This can be realized by altering the definition of $freq(p, \mathcal{D}')$ from $freq(p, \mathcal{D}') = count\{d : d \in \mathcal{D}' \wedge p \in d\}$ to $freq(p, \mathcal{D}') = \sum_{d \in \mathcal{D}' \wedge p \in d} rel(d)$, where $rel(d)$ is the relevance of d to the query. Thus, instead of counting the documents that contain p, we add their relevances to the query that determines $\mathcal{D}'$. This change does not affect the functionality of the mining algorithms. On the other hand, it may allow for techniques that examine only a subset of $\mathcal{D}'$ that is most relevant to the query, find the top-k interesting phrases in that subset only and obtain confidence bounds for their interestingness with respect to the complete $\mathcal{D}'$. Exploring this direction is left for future work.