Top-k Interesting Phrase Mining in Ad-hoc Collections using Sequence Pattern Indexing

Chuancong Gao, Sebastian Michel
Max-Planck Institute for Informatics   Saarland University
Saarbrücken, Germany
{cgao, smichel}@mmci.uni-saarland.de

ABSTRACT

In this paper we consider the problem of mining frequently occurring interesting phrases in large document collections in an ad-hoc fashion. Ad-hoc refers to the ability to perform such analyses over text corpora that can be an arbitrary subset of a global set of documents. Most of the times the identification of these ad-hoc document collections is driven by a user or application defined query with the aim of gathering statistics describing the sub-collection, as a starting point for further data analysis tasks. Our approach to mine the top-k most interesting phrases consists of a novel indexing technique, called Sequence Pattern Indexing (SeqPattIndex), that benefits from the observation that phrases often overlap sequentially. We devise a forest based index for phrases and an further improved version with additional redundancy elimination power. The actual top-k phrase mining algorithm operating on these indices is a combination of a simple merge join and inspired by the pattern-growth framework from the data mining community, making use of early termination and search space pruning technologies that enhance the runtime performance. Overall, our approach has on average a lower index space consumption as well as a lower runtime for the top-k phrase mining task, as we demonstrate in the experimental evaluation using real-world data.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications

General Terms

Algorithms, Experimentation, Performance

1. INTRODUCTION

Mining the essential characteristics of given document collections in the form of frequently occurring interesting phrases opens ground to a fine grasp of what is going on. The lessons that can be learned when inspecting such phrases are much beyond simple keyword based occurrence statistics w.r.t. their expressiveness. Examples of such phrases include person names, products (with their properties), locations, events, or even combinations of those. For large and hence often generic corpora, like the Web, however, it makes little sense to look at phrases extracted from the entire collection but rather focus on different facets, constricted to certain topics, locations, or different kinds of sources, such as blogs, micro-news (such as in Twitter), or newspaper articles, etc. These restrictions, referred to as queries in this paper, are not known a-priori and hence make an upfront pre-processing (indexing) of the sub-collection, induced by the query, impossible. Still, the nature of this data mining task calls for reasonably short response times as well as a highly compact index of the global document set.

In this paper we focus on solving this problem of discovering the top-k interesting phrases in ad-hoc document collections. Interestingness is usually measured by statistical methods that emphasize a phrase's discriminative power in the ad-hoc document collection compared to the global importance, instead of only looking at its local frequency. For example, in documents capturing the query "Apple", although the phrase "computer" is more popular than the phrase "MacBook", the latter should be considered as more interesting.

A viable approach to mine these top-k interesting phrases should refrain from reading a lot of data during the mining process, the overall runtime should be low, and the index (based on the entire set of documents) as compact as possible. In fact, as we will see, the index maintained by existing approaches leads to many unnecessary or redundant information accesses, caused by a sub-optimal index organization and causing a larger than necessary runtime performance.

Recently, Simitsis et al. [18] propose the *Phrase Inverted Indexing* algorithm that stores for each phrase a list of document (IDs) that contain the phrase. Interesting phrases are retrieved by intersecting the ad-hoc document collection's document ID list with each phrase's inverted list. However, this approach requires a full scan of all the phrases in the document corpus and is very costly. Although the authors also give a much faster approximate algorithm, its results are not always particularly accurate.

Bedathur et al. [6] devise a new algorithm called *Forward Indexing* to solve this problem of reading the entire set of inverted lists. Instead of storing each phrase's inverted list of document ID, Forward Indexing stores containing phrase ID lists (called forward list) for each document. Hence only forward lists of local ad-hoc documents need to be scanned instead of all documents in the corpus like the case of Phrase Inverted Indexing. To compute the top-k interesting phrases, a merge join is performed on the ad-hoc document collection. They also devise a very efficient early termination technology for the algorithm.

Since a global index needs to be built in advance to store each phrase's global information, the index size also becomes an crucial aspect of the devised systems. Besides Forward Indexing, [6] proposes another algorithm *Prefix-Maximal Indexing* to solve the problem, with a very compact index structure in memory of only up to half of the index size of Forward Indexing. This is achieved by storing only *prefix-maximal phrases* for each document, instead of all the containing phrase IDs. The prefix-maximal phrases form a very compact subset of all phrases, while covering all possible phrases in the document. The top-k interesting phrase computation of Prefix-Maximal Indexing also works like a merge join, joining each possible prefix at a time. However, due to the very complex merge join process and lack of available early termination techniques, Prefix-Maximal Indexing runs significantly slower, compared to Forward Indexing. As we can see in later sections, Prefix-Maximal Indexing can be hundreds times slower on large ad-hoc document collections.

Contribution and Outline

In this paper we propose a novel algorithm named *Sequence Pattern Indexing*, with the advantages of both Forward Indexing and Prefix-Maximal Indexing. By indexing prefix-maximal patterns via a novel carefully devised compact forest/graph structure, we avoid storing most of the duplicate sub-phrases, occurring in Prefix-Maximal Indexing. With a novel top-k interesting phrase computation method which combines both classic merge join and the pattern-growth framework [17], usually adopted by frequent pattern mining algorithms in data mining, together with new early termination and search space pruning technologies, we make our algorithm even times faster than Forward Indexing, on average. Evaluation results show that our algorithm has only about half the index size of Prefix-Maximal Indexing, and one fourth of Forward Indexing's, while performing 2 to 5 times faster than fastest Forward Indexing on average querying time (less than one second). We also evaluated the average number of bytes read, observing that our method has similar bytes read as the Forward Indexing technique, which is the lowest among evaluated algorithms.

This paper is organized as follows: Section 2 gives the detailed problem definition and introduces a toy example used in this paper for illustration purposes. Section 3 introduces the details of several previously proposed state-of-the-art algorithms solving the same problem. We provide our algorithm details in Section 4, including two indexing approaches and top-k interesting phrase computation algorithm. Evaluation details are given in Section 5, with dataset details, results on index size, results on average querying timeand results on average byte read. Finally Section 6 discusses related work, followed by the conclusion in Section 7.

2. PROBLEM DEFINITION

In this paper, we provide a novel and efficient algorithm to solve the problem of mining the top-k frequent interesting phrases (or patterns called in previous publications) in an ad-hoc document collection, usually corresponding to a query. More specifically, given a document corpus D and a query q, we would like to output at most k frequent phrases with the highest interestingness values in D', a subset of D with respect to q where each document $d \in D'$ satisfies query q. Given a document d which is a sequence over the word corpus, a phrase p is said to be contained by d if and only if p is a continuous sub-sequence of d, denoted by $p \sqsubseteq d$. A phrase's length is also restricted by user-specified minimum and maximum length thresholds ($min\text{-}len$ and $max\text{-}len$), with $1 \leq min\text{-}len \leq max\text{-}len \leq 6$. The frequency $|\{d|d \in D' \wedge p \sqsubseteq d\}|$ of a phrase p in a document collection D' is called the support value of p with respect to D', denoted by $sup_p^{D'}$ or just sup_p when in a clear context. A phrase is called frequent in a document collection D' if and only if $sup_p^{D'} \geq min\text{-}sup^{D'}$, where $min\text{-}sup^{D'}$ is a user-specified threshold. The interestingness measure we adopt here is the one most commonly used in previous work [18, 6], called confidence value (or somewhere relevance value) and is defined as follows.

DEFINITION 1. *Given a phrase p and an ad-hoc document collection D', the confidence value of p w.r.t. D' is defined as: $conf_p^{D'} = \frac{sup_p^{D'}}{sup_p^{D}}$, where D is the document corpus and $D' \subseteq D$. When D' is clear in the context, we can just use $conf_p$.*

Figure 1 gives a toy example with 4 documents in corpus D, together with the top-k ($k = 3$) interesting phrases.

ID	Document
d_1	⟨eabefcad⟩
d_2	⟨dcbeafe⟩
d_3	⟨fceacd⟩
d_4	⟨acfdbeacd⟩

D = $\{d_1, d_2, d_3, d_4\}$; D' = $\{d_2, d_3, d_4\}$

(a) Documents

ID	Phrase	Conf.
p_1	⟨ac⟩	2/2
p_2	⟨bea⟩	2/2
p_3	⟨ea⟩	3/4
p_4	⟨be⟩	2/3
......		

k = $\{p_1, p_2, p_3\}$

(b) Top-k Phrases

($min\text{-}sup^{D'} = 2, k = 3, min\text{-}len = 2, max\text{-}len = 4$)

Figure 1: Running Example

3. PREVIOUS ALGORITHMS

In this section, we will give a brief overview on existing techniques to this problem.

3.1 Phrase Inverted Indexing

Introduced in [18], this approach computes the top-k interesting phrases by creating an index storing all the inverted lists of phrases in the document corpus as follows. The inverted list of each phrase p contains the IDs of all the documents containing p. Now given an ad-hoc document collection D', the local support value of p ($sup_p^{D'}$) is computed by intersecting p's inverted list with the list of document IDs in D'. Since the global support value of p (sup_p^{D}) equals to the length of p's inverted list, the confidence value (interestingness) of p ($conf_p^{D'}$) with respect to D' is known. The top-k interesting patterns are selected by iterating over all the patterns' inverted lists. Figure 2 depicts the index structure on the document corpus D appeared in Figure 1.

ID	Document IDs
p_1	$\{d_1, d_2\}$
p_2	$\{d_1, d_3\}$
p_3	$\{d_1, d_3, d_4\}$
p_4	$\{d_1, d_2, d_3, d_4\}$

Figure 2: Phrase Inverted Indexing

ID	Forward List
d_1	$\{p_1{:}2, p_2{:}2, p_3{:}3, p_4{:}4\}$
d_2	$\{p_1{:}2, p_4{:}4\}$
d_3	$\{p_2{:}2, p_3{:}3, p_4{:}4\}$
d_4	$\{p_3{:}3, p_4{:}4\}$

Figure 3: Forward Indexing

However, this approach requires a full scan of all the inverted lists to compute the top-k phrases, which is very inefficient especially when there are, as in a typical corpus, millions of documents and phrases. Although [18] also introduces an efficient approximate method, its output is not guaranteed to contain the correct top-k results.

ID	Phrases
d_1	{ad, be, bef, befc, ca, cad, ea, ef, efc, efca, fc, fca, fcad}
d_2	{be, bea, cb, cbe, cbea, dc, dcb, dcbe, ea, fe}
d_3	{ac, acd, cd, ce, cea, ceac, ea, eac, eacd, fc, fce, fcea}
d_4	{ac, acd, acf, acfd, be, bea, cd, cf, cfd, ea, eac, fd}

......

(a) All Phrases

ID	Prefix-Maximal Phrases
d_1	{ad, befc, cad, ea, efca, fcad}
d_2	{bea, cbea, dcbe, ea, fe}
d_3	{acd, cd, ceac, eacd, fcea}
d_4	{acd, acfd, bea, cfd, eac, fd}

......

(b) Prefix-Maximal Phrases

ID	Maximal Phrases
d_1	{befc, ea, efca, fcad}
d_2	{cbea, dcbe, fe}
d_3	{ceac, eacd, fcea}
d_4	{acd, acfd, bea, eac}

......

(c) Maximal Phrases

Figure 4: Prefix-Maximal Indexing

3.2 Forward Indexing

Unlike the Phrase Invert Indexing method which stores the inverted list of each phrase, the recently proposed Forward Indexing approach [6] builds a forward list for each document $d \in D$, containing all the IDs of phrases contained in d. When trying to compute the top-k phrases of an ad-hoc document collection D', only the forward lists of documents in D' need be to scanned.

The phrase IDs in each forward list are stored in ascending order of their global support values. The top-k phrases are computed by performing a $|D'|$-way merge join. During this process, the local support value and further the confidence value of a phrase can be computed. The global support value for each phrase p is stored explicitly in the forward lists together with p's ID. Global support values could also be stored in a separate global dictionary. However, as mentioned in [6], storing them explicitly in the forward lists provides better performance. Figure 3 gives the index structure on document corpus D appeared in Figure 1, where numbers after ":" denote global support values.

In the implementation, a 64-bit integer is assigned for each phrase in a forward list, storing the phrase's global support value in the first 32 bits and the ID in the last 32 bits.

Although this approach is more efficient than Phrase Inverted Indexing, a full scan of D''s forward lists is still required. To improve its performance, [6] devises an early termination technique. For any phrase p, we have an upper bound $max\text{-}conf_p^{D'}$ of $conf_p^{D'}$:

$$max\text{-}conf_p^{D'} = \min\left\{1, \frac{|D'|}{sup_p^D}\right\} \geq \frac{sup_p^{D'}}{sup_p^D} = conf_p^{D'}$$

since $|D'| \geq sup_p^{D'}$. As the global support values are stored in the forward lists in ascending order, for any phrase p that precedes another phrase q, seen during the joining process, we know that $max\text{-}conf_p^{D'} \geq max\text{-}conf_q^{D'}$. Hence, we can safely stop the computing process if $max\text{-}conf_q^{D'}$ is not larger than the k-th phrase's confidence value in the top-k phrase list.

3.3 Prefix-Maximal Indexing

[6] also propose the Prefix-Maximal Indexing approach that indexes only prefix-maximal phrases. In contrast to the Forward Indexing approach which tries to accelerate the top-k interesting phrases computation, the Prefix-Maximal Indexing method reduces the memory requirement of the index structure. This is achieved by storing only prefix-maximal phrases (which is a very small subset of all phrases) with respect to each document in the corpus, instead of storing the IDs of all phrases contained by the document.

DEFINITION 2. *A phrase p is called prefix-maximal w.r.t. a document $d \in D$ iff (i) p is frequent in D (globally frequent), (ii) $p \sqsubseteq d$, and (iii) $\nexists p'$ such that p' is frequent in D, $p \sqsubset p' \sqsubseteq d$, and p is a prefix of p'.*

Storing only maximal phrases (a subset of prefix-maximal phrases derived by removing "p is a prefix of p'" in Definition 2) could further reduce the memory requirement, since all the non-maximal phrases contained by each document are also contained in the maximal phrases. However, the support values of some sub-phrases (those which are not prefixes of maximal phrases) can not be easily calculated using the Prefix-Maximal Indexing. Hence, prefix-maximal phrases are used instead.

For each document $d \in D$, the prefix-maximal phrases are stored in a forward list and ordered lexicographically. To further compress the index, for each prefix-maximal phrase only its suffix different from the previous phrase in the forward list is stored. For example, given the previous phrase $\langle$cd$\rangle$ in d_3 in Figure 4(b), only $\langle$ceac$\rangle$'s suffix $\langle$eac$\rangle$ needs to be stored. Support values of all the phrases are calculated by merging all the forward lists in D', comparing each phrase from a forward list to others. Over-counting of a sub-phrase is also avoided. Details of the merge algorithm can be found in [6]. After calculating all the local support values with respect to D', the confidence values are calculated by using a separate global support value dictionary. Then, the top-k results can be outputted. Figure 4 presents a comparison between storing all phrases, only maximal phrases, and prefix-maximal phrases. Note that here the document corpus D is no longer limited to only the 4 given documents in Figure 1, but with many other documents not listed ($|D| \gg 4$). Hence, many of the sub-phrases are globally frequent. In the following, if not mentioned, we have $|D| \gg 4$.

To store the index more efficiently, all the prefix-maximal phrases contained by one document are merged together forming a long integer array. A 32-bit integer is assigned for each item, while only the last 24 bits are used for content. The first 8 bits of the first item in each phrase contains a flag indicating the start of a new phrase, and stores the difference length from the previous phrase. Note that this is an improved implementation, as the original implementation of Prefix-Maximal Indexing stores each prefix-maximal phrase in an individual array, and has to also keep array addresses (with the size of 64 bits in our X64 systems) of all the prefix-maximal phrases. On a large scale document corpus with average long document length, a large part of memory would be used for storing array addresses.

The main drawback of this approach is that a full scan of the forward lists of D' is required, since no early termination technique can be applied. Experiments show that this approach is very slow compared to the Forward Indexing method. However, both of them are dramatically faster than Phrase Inverted Indexing.

4. OUR APPROACH

In this section we present our Sequence Pattern Indexing approach for indexing and mining top-k phrases. First, we present the details of our compressed global phrase index in Section 4.1 followed by its improvement in Section 4.2. Then, Section 4.3

presents the algorithm to compute the top-k interesting phrases on the given ad-hoc document collection.

4.1 Sequence Pattern Indexing

Although storing only prefix-maximal phrases could reduce the memory usage significantly, compared to storing all the phrase IDs, it is still not perfect. As an illustration, consider the prefix-maximal phrases in Figure 4(b): we can find a lot of duplicate sub-phrases, like $\langle\texttt{c}\underline{\texttt{eac}}\rangle$ and $\langle\underline{\texttt{eac}}\texttt{d}\rangle$ in d_3, $\langle\texttt{e}\underline{\texttt{fca}}\rangle$ and $\langle\underline{\texttt{fca}}\texttt{d}\rangle$ in d_1, etc. These duplicate sub-phrases occupy a large amount of memory, in particular with low $min\text{-}sup^D$ and large $max\text{-}len$. Our devised index structure makes use of this observation to achieve a better compression by removing these duplicate sub-phrases, as many as possible. Obviously, the new indexing technique should also not jeopardize the querying performance of the top-k phrase computation. As we will see in Section 5, our indexing and top-k phrase mining methods provide better performance on both the index size and the running time compared to existing approaches.

Our indexing approach is inspired by the following observation: Phrases are ordered sequences, which are originally extracted from a long ordered sequence of the document. Hence, they also implicitly reflect an order between themselves. If we sort all the prefix-maximal phrases by their original matching positions in the document as shown in Figure 5 (Numbers after "@" indicate matching positions.), we observe that there are many of duplicates as a suffix of the previous phrase, while being a prefix of the current phrase. In order to achieve a smaller index size, we need to find a way to utilize the original matching positions (or order) of prefix-maximal phrases more efficiently. One simple way is keeping, for each phrase, only the rest part after the duplicate prefix. However, this could not compress prefix-maximal phrases with the same prefix (e.g., document $\langle\texttt{ababc}\rangle$ with its only two prefix-maximal phrases $\langle\texttt{aba}\rangle$ and $\langle\texttt{abc}\rangle$.) which happens a lot in real datasets (though not very obvious in our tiny toy example). Plus, it is very difficult to locate all possible suffixes of a specific prefix phrase, which is very important in top-k interesting phrase computation. Note that similarly, Prefix-Maximal Indexing does store only the difference suffix for previous phrase sharing the same prefix, it is under the assumption that all phrases are ordered lexically, where phrases with the same prefix are grouped together. Also, that approach is significant slow in top-k interesting phrases computation, due to the highly complex computation on possible prefix phrases.

ID	Prefix-Maximal Phrases [a]
d_1	$\{\boxed{\texttt{e}}\texttt{a}@1, \texttt{befc}@3, \underline{\boxed{\texttt{e}}\texttt{fc}}\texttt{a}@4, \underline{\texttt{fca}}\texttt{d}@5, \underline{\texttt{cad}}@6, \underline{\texttt{ad}}@7\}$
d_2	$\{\texttt{dcbe}@1, \underline{\texttt{cbe}}\texttt{a}@2, \underline{\texttt{bea}}@3, \underline{\texttt{ea}}@4, \texttt{fe}@6\}$
d_3	$\{\texttt{fcea}@1, \underline{\texttt{cea}}\texttt{c}@2, \underline{\texttt{eac}}\texttt{d}@3, \underline{\texttt{acd}}@4, \underline{\texttt{cd}}@5\}$
d_4	$\{\boxed{\texttt{ac}}\texttt{fd}@1, \underline{\texttt{cfd}}@2, \underline{\texttt{fd}}@3, \texttt{bea}@5, \underline{\texttt{ea}}\texttt{cd}@6, \underline{\boxed{\texttt{ac}}\texttt{d}}@7\}$

......

[a] $\underline{\quad}$ – Duplicate Prefix Sub-Phrase Comparing to Previous Phrase
$\boxed{}$ – Duplicate Prefixes among Phrases

Figure 5: Prefix-Maximal Phrases (Ordered by Matching Position)

Figure 6(a) gives the data structure of our new indexing method on document d_1's 6 prefix-maximal phrases given in Figure 5. We can see that all the sub-phrases are organized on a forest structure (two trees in the forest in this example), where each node contains one item and its possible length information (will be explained later). There are also some links from outside pointing to the nodes (drawn in arrows above the nodes), which indicate the starting item of the indexed prefix-maximal phrases. These information together with the item in each node are used to store phrase length information, which is a bit array with the size of $max\text{-}len$. Its flag value i indicates that a phrase with the item as the i-th item is valid.

For example, if we want to retrieve all the phrases starting from item b, we first visit the unique node n (n_3 as shown in Figure 6(a)) which is linked from outside and contains b, and we have a prefix $p = \langle\texttt{b}\rangle$ with the length of 1. Since n has only one child, we can only extend it with e. We can keep extend p until $\langle\texttt{befc}\rangle$ which is still valid. When we are trying to extend it with the next node containing a which results in a prefix with the length of 5, we find that it is no longer valid since the next node does not contain the possible length information about length 5 and we have to stop here. In this way, all the phrases started with b can be visited.

The index is built as follows:

(I) For each prefix-maximal phrase p from a document d's prefix-maximal phrases ordered by matching position, we check whether p's first item $p[1]$ has been linked to the index. If not, we create a new tree root node n with $p[1]$ as its content and add 1 to n's length information.

(II) We add p's next item $p[i]$ to the tree as a child of node n. If it is already there we just skip it, else we create a new node for $p[i]$. Possible length information for $p[i]$ are also updated with i added. Node n is also set to its child node contains $p[i]$. Then we increase i by 1 and repeat (II).

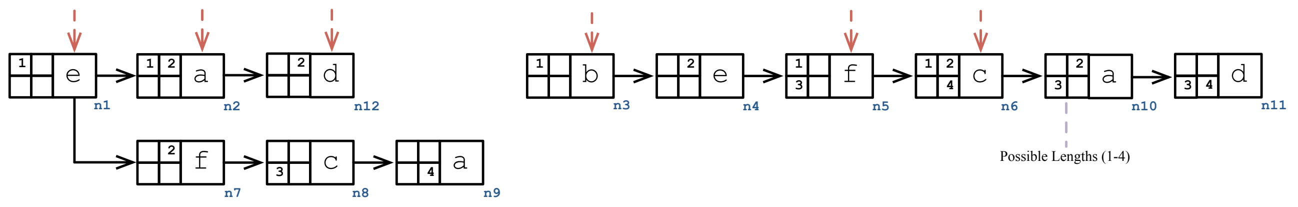
(III) Until all of p's items have been added, we go back to (I) for the next prefix-maximal phrase. All the items we have not seen before are also linked from outside to the index during this process. The node ID in right-bottom corner of each node in Figure 6(a) denotes the order each node is created.

Note that during the index building each unique item gets an outside link pointing to it when it was encountered the first time, and this may also include some outside links to items which are not starting items of any phrases (especially when $min\text{-}len \geq 2$). For example, one may notice there is one arrow on item d in Figure 5, while there is no phrase which starts with d (except length-1 phrase d which is not valid since $min\text{-}len = 2$). This is because that when we first encounter an item we do not know whether there are more items afterward. We can either remove the outside link by doing a simple checking after index building, or simply ignore it as the possible length information guarantees no invalid phrase will be retrieved with the item as starting item.

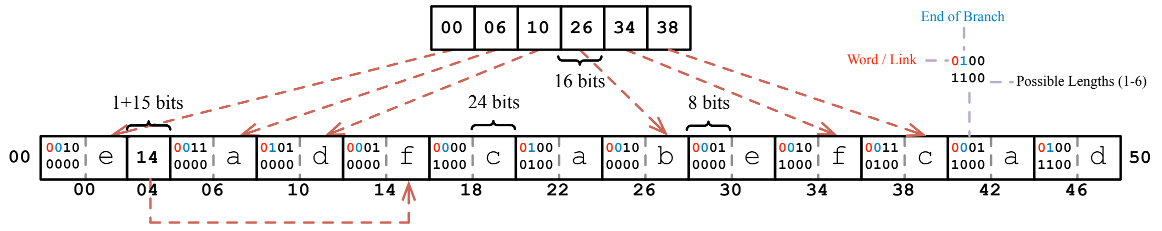
We can further prove its correctness with the following theorem.

THEOREM 1. *Given an index generated from a document d's prefix-maximal phrases using the Sequence Pattern Indexing approach, each possible length information stored in each node in the index corresponds to a different phrase contained by d.*

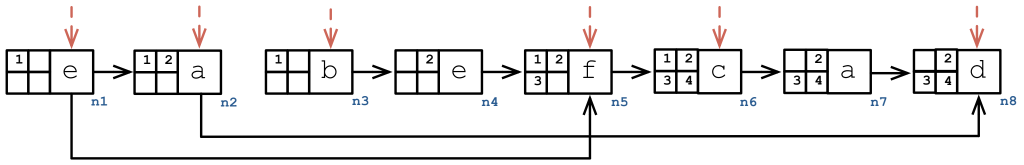
PROOF. During the indexing process sub-phrases from different prefix-maximal phrases are stored on different parts of the forest due to their different prefixes and all of the sub-phrases of a prefix-maximal phrase are indexed with different possible lengths set. Hence, the number of possible lengths equals the number of phrases. Since our index structure is a forest, given a node n and one of its possible lengths l, we can always get a different phrase p with the length of l by traversing n and its $l - 1$ ancestors. Recall that each possible length information is set only when indexing one of the prefix-maximal phrase p', we have $p \sqsubseteq p'$. Now we have x different phrases where each phrase is a sub-phrase of d and x equals the total number of phrases of d, and we prove this theorem. □



(a) Index Structure



(b) Memory Layout



(c) Improved Index Structure

Figure 6: Sequence Pattern Indexing (On $d_1 \in D$ in Figure 1)

From Theorem 1 the following corollary can be derived.

COROLLARY 1. *By traversing the index of a document d generated by Sequence Pattern Indexing, we can get the exact set of phrases contained by d.*

PROOF. Theorem 1 has proved for each prefix-maximal phrase we have all its sub-phrases indexed. According to previous description about index visiting process, given any prefix phrase we can visit all its super-phrases with the prefix phrase as prefix. Together, the corollary is proved. □

As we can see, building the index of each document is with only $O(n)$ time complexity (as items in each phrase are visited only once), where n is the sum of lengths of all the prefix-maximal phrases contained by the document. Hence, the index building is with quite low cost, and on our evaluaiton it actually finishs in only a few minutes with millions of documents.

Figure 6(b) depicts the memory layout of our indexing technique, with the number below each cell indicating a relative memory address. As shown in the figure, every branch in the forest are merged together to a byte array. Each node is assigned with 4 bytes (32 bits), with the first byte (8 bits) used for storing flags, and the last 3 bytes (24 bits) storing the item content. In the flag byte, the last 6 bits are kept for possible length (1 to 6) information (given $max\text{-}len \leq 6$ in our systems) mentioned before. The second bit is used to indicate the end of a branch which is set when merging the branches to generate the byte array. With that flag set to 1, the phrase matching always stops right after that node. When set to 0, the first bit indicates that this 4 bytes belong to a node. When it is set to 1, the current 2 bytes (with 15 bits left now after the first flag bit) are used for storing a link address to a child node. Outside links to the start items are also organized into a list with 2 bytes containing each item's address. As we can see, the outside link list does not contain any information about the item except their addresses. This will be explained later in Section 4.3.

4.2 Improved Sequence Pattern Indexing

Although compared to Prefix-Maximal Indexing, the memory usage has been greatly reduced by using the Sequence Pattern Indexing, there exist still some duplication in the forest structure, like $e \to f \to c \to a$ and $a \to d$ depicted in Figure 6(a). Our first indexing method above can not deal with them, as it utilizes only ordered prefix-maximal phrases. Also, some of the duplicates in index structure are not duplicate sub-phrases. Hence, we further improve our Sequence Pattern Indexing approach, by utilizing the specific matching positions (instead of just matching orders) of each prefix-maximal phrases, trying to eliminate more duplicates.

Figure 6(c) shows our improved data structure. Compared to the original one in Figure 6(a) using a forest structure, we now make use of a graph structure which can eliminate duplicates more efficiently. While the traversing method remains basically unchanged, the index building process is substantially modified:

(I) For each prefix-maximal phrase p together with its matching position m from all the document d's prefix-maximal phrases ordered by matching position, we first check whether $p[1]$ has been linked. If not, we create a new node n from $p[1]$.

(II) For each p's next item $p[i]$, we check whether $p[i]$ matches a node n_{m+i} whose matching position equals $m+i$. If yes, we increase i, link and set n to n_{m+i}, and repeat (II). If n_{m+i} does not exist, we create a new node.

(III) Until all of p's items have been added, we go back to (I) for the next prefix-maximal phrase. Possible length information are updated as before. Node numbers in the right-bottom corners in Figure 6(c) now indicate both the adding order and the matching positions.

For example, when adding the third phrase $\langle$`efca`$\rangle$ whose matching position is 4, we first match the first item `e` to node n_1. Since n_1's next node dose not have a matching position of 5, we link n_1 to node n_5 whose matching position is 5 and contains the item `f`, and continue matching on node n_5 until all the items are added.

Note that now when we are traversing the index structure, only the direct links after the first phrase items are followed, to prevent retrieving false phrases. For example, when we are retrieving phrases starting with `e`, we only follow the direct link directly after `e` linking to `d`, but not the indirect one after the next item `a`, which could lead to a false phrase $\langle$`ead`$\rangle$. This is because if we follow the indirect links there may be multiple paths (phrases) with the same length but different contents to a node, and we do not know which path the matching possible length information of the node belongs to and hence is valid. For example if we follow the indirect links, both the false phrase $\langle$`ead`$\rangle$ and the true phrase $\langle$`cad`$\rangle$ would lead to node n_8 with the same possible length of 3 in Figure 6(c).

Like Sequence Pattern Indexing, we also prove the correctness of the improved approach. Before we continue, we first prove one lemma.

LEMMA 1. *Given an index generated by Improved Sequence Pattern Indexing, each node can have at most two parents and they have the same content.*

PROOF. Given a document d and a node n with the content $d[m]$ and its matching position m, there is only one possible item $d[m-1$ before $d[m]$ in document d at position $m-1$. Besides n's trivial parent node n' with the matching position $m-1$ (if n' exists), according to the index building process another parent node n'' could only link to n if and only if n'''s content equals to $d[m-1]$. Now since when building the index there is only one phrase corresponding to this linking behavior, there is at most one parent that links to n. Together with the trivial parent node, n could only have two parents with the same content, and the lemma is proved. □

THEOREM 2. *Given an index generated from a document d's prefix-maximal phrases and their matching positions using the Improved Sequence Pattern Indexing approach, each possible length information stored in each node in the index corresponds to a different phrase contained by d.*

PROOF. We only need to prove that for a node n with multiple parents this theorem is true. For n's descendants the same proof can be applied. While for those nodes with single parent Theorem 1 in Section 4.1 has already proved it.

Since we have already proved that n could only have two parents n' and n'' with the same content in Lemma 1, obviously the two paths from n' and n'' to n are same and corresponds to one same phrase, according to the index building process and the modified traversing process of following only direct links after the first item. Hence it is just like the single parent situation and we have that for a node with multiple parents each possible length information corresponds to a different phrase. □

From Theorem 2 we can derive the following corollary.

COROLLARY 2. *By traversing the index of a document d generated by Improved Sequence Pattern Indexing, we obtain exactly those phrases contained by d.*

PROOF. Very similar to the proof of Corollary 1. □

Similarily, the improved index building approach is also with only $O(n)$ time complexity.

The technique used to build the index memory layout is exactly the same one for building Sequence Pattern Indexing in Section 4.1.

4.3 Top-k Interesting Phrase Mining

Now, we specify the algorithm for computing the top-k interesting phrases, given the built index on the document corpus. Unlike the Forward Indexing approach and the Phrase-Maximal Indexing approach which use a $|D'|$-way merge join or variations, our algorithm (Sequence Pattern Indexing) is a combination of both $|D'|$-way merge join and the pattern-growth framework [17] used in the frequent sequential pattern mining problem in data mining. While the merge join approach is used only on prefixes with length 1, the pattern-growth framework is applied to extend each prefix in the merge join process.

The Pattern-growth framework in essence is a depth-first search algorithm, where during each step the prefix pattern is extended with one of the new local items derived by scanning the rest unmatched part of each instance containing the prefix pattern. Algorithms adopting pattern-growth framework usually start with an empty pattern, and guarantee each newly derived prefix pattern is frequent by checking the minimum support threshold. In this way, all frequent patterns can be discovered and visited. For example, if each prefix pattern is extended with one of the respective local items in lexicographical order, we could have patterns found in the following order: $\langle$`a`$\rangle$, $\langle$`ab`$\rangle$, $\langle$`aba`$\rangle$, $\langle$`abd`$\rangle$, $\langle$`b`$\rangle$, $\langle$`ba`$\rangle$, $\langle$`bd`$\rangle$, $\langle$`cd`$\rangle$, $\langle$`d`$\rangle$, $\langle$`db`$\rangle$.

When we introduced the index structure of the Sequence Pattern Indexing method in Section 4.1, we pointed out that each index for each document contains an address list consisting of addresses of different items in the index. If we sort each address list in advance, when building the index, by keeping the same order of all the items linked-to, among all the documents in the corpus, we can actually apply the $|D'|$-way merge join on those address lists, and retrieve all the phrases with the length of 1 appearing in the ad-hoc document collection D'. Now, given any prefix length 1 phrase p and the index structures of each document in D', we can apply the pattern-growth framework on p by keep appending local frequent items discovered in the index structures to p, to mine all the phrases with p as prefix and update the top-k interesting phrase list.

Early termination can also be achieved during the $|D'|$-way merge join process. Before going into details, we first give some definitions and theorems.

DEFINITION 3. *Given a phrase p, we use:* $min\text{-}des\text{-}sup_p = \min\{sup_{p'} | p' \textit{ is frequent } \wedge\ p \textit{ is a prefix of } p'\}$ *to denote the minimum frequent support value of p's super-phrase p' with p as prefix (p can be equal to p').*

THEOREM 3. *Given a length 1 phrase p and any super-phrase p' with p as prefix (p can be equal to p'), we have:*

$$\begin{aligned} max\text{-}conf_p^{D'} &= \min\left\{1, \frac{|D'|}{min\text{-}des\text{-}sup_p^D}\right\} \\ &\geq \min\left\{1, \frac{sup_p^{D'}}{min\text{-}des\text{-}sup_p^D}\right\} \\ &\geq \frac{sup_{p'}^{D'}}{sup_{p'}^{D}} = conf_{p'}^{D'} \end{aligned}$$

as an upper-bound of both p and p''s confidence values.

PROOF. Given the famous Apriori Principle [4] proving that sup_p always $\geq$ $sup_{p'}$ and the fact that $|D'| \geq sup_p^{D'}$, this theorem can be easily proved. □

Now we can derive from Theorem 3 the following corollary.

COROLLARY 3. *Given a list of length-1 phrases $p_1, \cdots, p_i, \cdots, p_n$ sorted by $min\text{-}des\text{-}sup_{p_i}$ in ascending order, we have: $max\text{-}conf_{p_1}^{D'} \geq max\text{-}conf_{p_i}^{D'} \geq \cdots \geq max\text{-}conf_{p_n}^{D'}$*

PROOF. For any length-1 phrases p_i and p_j with $min\text{-}des\text{-}sup_{p_i} \leq min\text{-}des\text{-}sup_{p_j}$, we always have:

$$\begin{aligned} max\text{-}conf_{p_i}^{D'} &= \min\left\{1, \frac{|D'|}{min\text{-}des\text{-}sup_{p_i}^{D}}\right\} \\ &\geq \min\left\{1, \frac{|D'|}{min\text{-}des\text{-}sup_{p_j}^{D}}\right\} \\ &= max\text{-}conf_{p_j}^{D'} \end{aligned}$$

, and the corollary is proved. □

Now with Corollary 3, early termination can be easily applied. Before mining a length 1 phrase p from the $|D'|$-way merge join, we check whether $max\text{-}conf_p^{D'}$ is not larger than the k-th result in the top-k list, and stop safely if it is true. Length-1 phrases of each documents are retrieved in the order of $min\text{-}des\text{-}sup_{p_i}$ from the address list in each index of each document, by sorting all items in the corpus in advance during the index building process.

We also give some theorems for search space pruning before extending any super-phrases of each length 1 phrase. Similar to the proof of Theorem 3, we can prove the following theorem.

THEOREM 4. *Given a phrase p and any super-phrase p' with p as prefix (p can be equal to p'), we have:*

$$\begin{aligned} max\text{-}des\text{-}conf_p^{D'} &= \min\left\{1, \frac{sup_p^{D'}}{min\text{-}des\text{-}sup_p^{D}}\right\} \\ &\geq \frac{sup_{p'}^{D'}}{sup_{p'}^{D}} = conf_{p'}^{D'} \end{aligned}$$

as a upper-bound of both p and p''s confidence values.

From Theorem 4 we get to the following corollary.

COROLLARY 4. *Given a prefix phrase p and its length-$(|p|+1)$ super-phrases with p as prefix and extended with p's local frequent items, we can safely prune each length-$(|p|+1)$ phrase p' by checking whether $max\text{-}des\text{-}conf_{p'}^{D'}$ is not larger than the k-th result in the top-k list.*

PROOF. Since for p and all its super-phrases, the upper bound their confidences is always not greater than the last entry the top-k list, the top-k list would never be updated by p and its super-phrases. Hence, there is no need to extend p, and we can safely remove p and its super-phrases from the search space. □

Although very similar, $max\text{-}des\text{-}conf_p^{D'}$ is different (smaller and closer to $conf_p^{D'}$) from $max\text{-}conf_{p'}^{D'}$, since in the $|D'|$-way merge join D' and also $sup_p^{D'}$ are unknown until we meet (also necessary for applying early termination) p, while in pattern-growth $sup_p^{D'}$ is already computed before p's local frequent item computation.

Actually, we can also apply early termination locally when extending a phrase. The following corollary gives the details.

COROLLARY 5. *Given a prefix phrase p and its length-$(|p|+1)$ super-phrases with p as prefix and extended with p's local frequent items, ordered by $max\text{-}des\text{-}conf_{p'}^{D'}$ (p' denotes each of the super-phrases) in descending order, we can safely stop extending p iff for any p' its $max\text{-}des\text{-}conf_{p'}^{D'}$ is not larger than the k-th result in the top-k list.*

PROOF. Similar to the proof of Corollary 4. □

Algorithm 1 shows the detailed mining algorithm. $entries_d$ is used to denote the index's address list of each document $d \in D'$, while $pdb_p^{D'}$ denotes the projected database w.r.t. p on D' containing each matching position of p on D'. The local frequent items in $pdb_p^{D'}$ are calculated by traversing the index structure of each d from the p's last matched item with only frequent ones returned. The top-k result list is updated each time a new prefix phrase p with $min\text{-}len \leq |p| \leq max\text{-}len$ is visited, by comparing its confidence value $conf_p^{D'}$ with others' in top-k result list.

Algorithm 1: Top-k Phrase Mining (Sequence Pattern Indexing)

```
Function: $mine(D', k)$
Input: Current Document Collection $D' \subseteq D$, Result Number
Output: Top-$k$ Interesting Phrase List
1 $result \leftarrow \emptyset$;
2 foreach frequent item i from $|D'|$-way merge join on $entries_d$
  of each $d \in D'$ do
3     $p \leftarrow \langle i \rangle$;
4     if $result[k] \geq max\text{-}conf_p^{D'}$ then
5         break;
6     $mine\_local(D', k, p, pdb_p^{D'}, result)$;
7 return $result$;

Function: $mine\_local(D', k, p, pdb_p^{D'}, result)$
Input: Current Document Collection $D'$, Result Number,
       Current Prefix Phrase, Current Projected Database,
       Top-$k$ Result List
1 if $min\text{-}len \leq |p| \leq max\text{-}len$ then
2     update $result$ with $p$ by comparing $conf_p^{D'}$;
3 foreach local frequent item i in $pdb_p^{D'}$, sorted by
  $max\text{-}des\text{-}conf_p^{D'}$ in descending order do
4     $p' \leftarrow p + \langle i \rangle$;
5     if $result[k] \geq max\text{-}des\text{-}conf_p^{D'}$ then
6         break;
7     $mine\_local(D', k, p', pdb_{p'}^{D'}, result)$;
```

5. EXPERIMENTS

We now report on an experimental evaluation of our devised SeqPattIndex (Sequence Pattern Indexing) approach. We compare its performance to two state-of-the-art methods: ForwardIndex (Forward Indexing) and PreMaxIndex (Prefix-Maximal Indexing). All experiments are conducted on a Dell Workstation with a quad core Intel Xeon W3520 (2.66 GHz) CPU (with only one core used) and 24 GB memory installed, running Windows 7 64-bit edition. All the algorithms are written by C#. Table 1 gives the algorithms' short names and descriptions for reference.

First, we introduce the two datasets we used during the evaluation, which are extracted from PubMed [3] – the largest free publication database on life sciences and biomedical topics. The first

Table 1: Algorithms Used in Evaluation

Baselines	
Algorithm	Description
ForwardIndex	Forward Indexing [6] (in Section 3.2)
PreMaxIndex	Prefix-Maximal Indexing [6] (in Section 3.3)

Our Approaches	
Algorithm	Description
SeqPattIndex	Sequence Pattern Indexing (in Section 4)
SeqPattIndex_IM	Improved Sequence Pattern Indexing (in Section 4.2)
SeqPattIndex_NP	Sequence Pattern Indexing with Early Termination and Search Space Pruning Turned off

one we used is the whole set of nearly 18 million publication titles (with a raw size of 1.5 GB), which has a relatively short average document length and very few duplicate words in each document. This kind of data is very common, from publication titles in academic databases like Google Scholar, DBLP [1] [1] to query logs, short messages as in Twitter. The second one we adopted is a set of 2.5 million publication abstracts (with a raw size of 2.4 GB), which has a relatively long average document length and a lot of duplicate words in each document. This kind of data is also very common, like in patent abstracts or NSF Award [2] abstracts, lead sections [2] on Wikipedia pages, or product descriptions on websites. The detailed dataset characteristics are listed in Table 2.

Table 2: Dataset characteristics

	Global		Per Document		
Dataset	#Doc.	#Word	Max. Len.	Avg. Len.	Avg. #Word
PubMed Titles	17,826,927	2,028,673	167	11.59	10.99
PubMed Abstracts	2,500,000	2,075,526	1599	149.72	92.1

5.1 Offline Global Frequent Phrase Mining

Note that we need to mine global frequent sub-phrases in advance on the whole dataset. Mined sub-phrases will then be indexed by the evaluated algorithms. We give the details of our adopted offline mining algorithm in this section.

A lot of algorithms have been devised to mine generalized frequent sequential patterns allowing varying length and gaps between items, like PrefixSpan [17]. However, those kind of algorithms are not suitable for our task of mining frequent continuous sequential patterns with maximum and minimum length limits and no gap between items. Although the length limitation can be easily solved, the continuous limitation is not easy to overcome. For example, one may consider using a general mining algorithm like PrefixSpan to mine all possible patterns first and then filter out our desired patterns. However, this approach is not only significantly slow but also inaccurate, as in some cases some continuous patterns could not be found. Following we give an example of the case:

Document:	abcabcd
Actually Pattern:	abcabcd
Found by PrefixSpan:	abcabcd
Found by PrefixSpan (No-gap):	abcabcd

As we can see, although PrefixSpan also can find a pattern with the content $\langle \texttt{abcd} \rangle$, the one it finds is non-continuous and will be filtered out, instead of the missed continuous one appearing later in the document. Although we can also add the no-gap limitation to PrefixSpan by extending a current pattern with only the local frequent items right after it when mining patterns, as shown in the example, we can only find the continuous pattern $\langle \texttt{abc} \rangle$ instead of $\langle \texttt{abcd} \rangle$ which is extended from the second appearance of $\langle \texttt{abc} \rangle$, since, according to the pattern-growth framework adopted by PrefixSpan, only the first appearance of pattern $\langle \texttt{abc} \rangle$ is considered.

Instead, we adopt a tricky way to help mining our desired patterns, by modifying the SeqPattIndex algorithm in Section 4.3 from mining the locally top-k interesting patterns to mining global frequent patterns. This conversion is simple, since we just need to remove all the early termination and search space pruning techniques and the top-k pattern list and instead output all the patterns that are frequent. In the following, we give the pseudo-code of the modified algorithm. Note that before we run the algorithm, we first create an index by treating each sub-text in a document with the length of at most $max\text{-}len$ as a prefix-maximal pattern. For example, the document $\langle \texttt{abcabcd} \rangle$ in the previous example will be indexed and mined by splitting into the following 7 prefix-maximal patterns $\{\langle \texttt{abca} \rangle, \langle \texttt{bcab} \rangle, \langle \texttt{cabc} \rangle, \langle \texttt{abcd} \rangle, \langle \texttt{bcd} \rangle, \langle \texttt{cd} \rangle, \langle \texttt{d} \rangle\}$ with $max\text{-}len$ of 4.

Algorithm 2: Global Frequent Phrase Mining

```
Function: $mine(D)$
Input: Global Document Collection
1 foreach frequent item i from $|D|$-way merge join on $entries_d$ of each $d \in D$ do
2     $p \leftarrow \langle \texttt{i} \rangle$;
3     $mine\_local(D, k, p, pdb_p^D)$;

Function: $mine\_local(D', p, pdb_p^{D'})$
Input: Current Document Collection $D'$, Current Prefix
       Phrase, Current Projected Database
1 if $min\text{-}len \leq |p| \leq max\text{-}len$ then
2     Output $p$;
3 foreach local frequent item $i$ in $pdb_p^D$ do
4     $p' \leftarrow p + \langle \texttt{i} \rangle$;
5     $mine\_local(D', p', pdb_{p'}^{D'})$;
```

The reason our algorithm can find all patterns is that each possible pattern with at most the length $max\text{-}len$ has already been indexed before running the algorithm, and will be visited if it is frequent.

Since only continuous patterns are mined and no time-costly filtering step is involved, this method is very fast. As we tested, only less than one hour is need for mining patterns on the two PubMed datasets (mentioned in Section 5) with $min\text{-}sup$ of 5 and $max\text{-}len$ of 6. Together with the time for buildig indexes using mined global patterns which is only a few minutes, the whole offline indexing step uses not more than one hour.

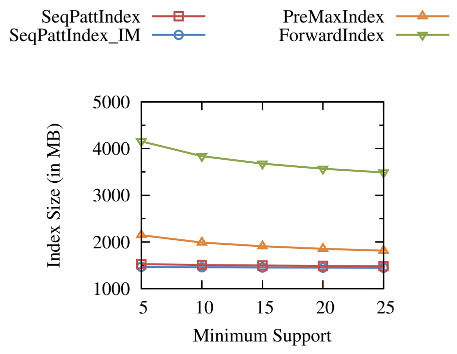
5.2 In-Memory Index Size Evaluation

We first evaluated the in-memory index sizes of our approaches (including the improved approach introduced in Section 4.2 called SeqPattIndex_IM here) and the two baselines, with the results under different minimum support thresholds listed in Figure 7. As we can see, Forward Indexing has a very large index size compared to others, while Prefix-Maximal Indexing has a very compact index size which is only half of that generated by Forward Indexing. Our methods have the best performances, with $1/2$ to $2/3$ of the Prefix-Maximal Indexing index size. On the PubMed Titles dataset, our improved approach has similar performance compared

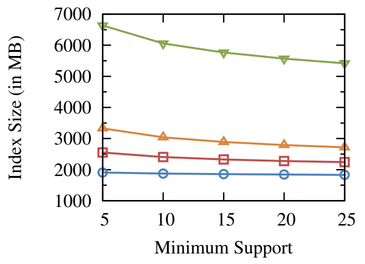
[1]We do not use DBLP as it contains only about 2 million publication titles and is too small to show significant runtime performance differences. It also does not include any publication abstracts.

[2]The section before the table of content and the first heading.

to the original one, since very few duplicate items can be found in each document in the dataset, hence, SeqPattIndex_IM could not further compress the index. On the PubMed Abstracts dataset, with lots of duplicate items, the improved approach further reduces the index size to $2/3$ to the original one's. Note that the index size of SeqPattIndex_NP is the same as for SeqPattIndex (i.e., SeqPattIndex_NP affects only the querying performance.).



(a) PubMed Titles



(b) PubMed Abstracts

Figure 7: In-Memory Index Size (in MB)

5.3 Querying Time Evaluation

Now we discuss the evaluation of the average runtime needed to compute the top-k interesting phrases. We selected the queries used in this evaluation as follows: First, we divide all sub-phrases into different groups based on their frequencies in the original global document collection. Patterns with the same number of digits in their frequencies are grouped together. We build ranges of frequencies like $[100, 1000)$, $[1000, 10000)$, etc. Then, we select the top-10 frequent sub-phrases in each such group and merge them together as the final evaluation query collection for computing the average querying time (runtime). The reason for selecting the queries like this is to cover different situations. Note that we do not consider queries whose frequencies are below 100 since their querying time with respect to different algorithms are too small to be distinguished and compared. Since the improved approach introduced in Section 4.2 has nearly the same querying time of the original approach (in Section 4.1), we do not show its results here. We also included the performance of SeqPattIndex by turning off all the early termination and search space pruning functions, indicated by SeqPattIndex_NP in the figures later. Note that the evaluation here assumes the indexes have alraedy been built and loaded in the memory, hence the results do not include runtimes on global pattern mining, maximal pattern extraction, index building, etc., in the previous offline step.

Figure 8(a) and 8(d) reports on the average querying time on PubMed Titles and PubMed Abstracts, respectively, with varying ad-hoc document collection size $|D'|$, a fixed minimum support threshold $min\text{-}sup = 5$, and $k = 10$. We compare our algorithm SeqPattIndex with the two state-of-the-art algorithms ForwardIndex and PreMaxIndex. As we can see, for all the algorithms, the average querying times increase as $|D'|$ increases with approximately a linear relationship. Note that when $|D'| > 10000$ the two algorithms SeqPattIndex and ForwardIndex start to show better performance, since their early termination technologies start to work more often under a larger $|D'|$ which results in a significantly reduced running time. Besides when $|D'|$ is not large even SeqPattIndex's slowest version SeqPattIndex_NP shows better performance than ForwardIndex. This is because ForwardIndex has to traverse all the frequent and infrequent sub-phrases when early termination is not working well, while SeqPattIndex only traverses those frequent sub-phrases whose parents are also frequent (since it follows the Apriori Principle [4]) and saves lot of time. Note that only queries with a frequency larger than $|D'|$ are adopted with the first $|D'|$ documents for each query used.

Figure 8(b) and 8(e) show the evaluation results with varying k. Queries with frequencies larger than k are used in the evaluation. As we can see, the querying time increases with increasing k. When k is very large (close to $|D'|$), SeqPattIndex tends to have the same average querying time as ForwardIndex. Note that although according to the algorithm in Section 3.3 PreMaxIndex should have the same performance with respect to all k values, actually its average querying time grows slowly with larger k. This is because of the slower update process of the top-k prefix list with a larger k.

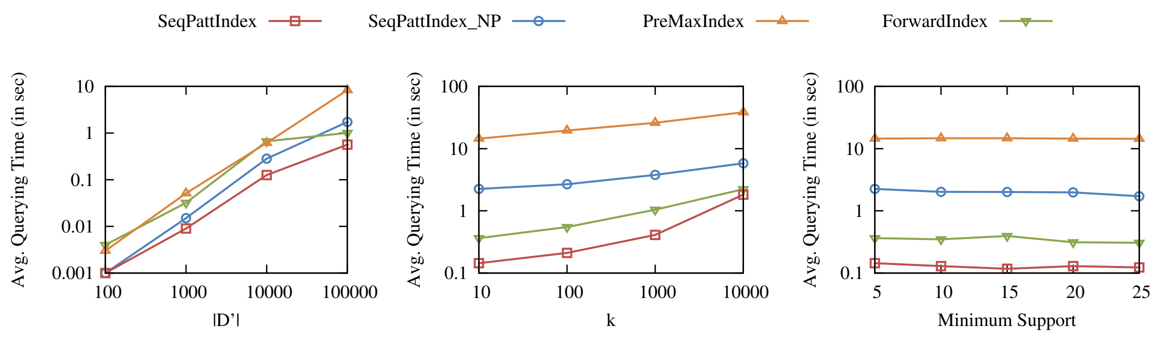
Finally, Figure 8(c) and 8(f) show the evaluation results on different minimum support thresholds $min\text{-}sup$. As we see, all the algorithms are not sensitive to $min\text{-}sup$.

In general, Prefix-Maximal Indexing shows the worst performance in all the cases and is slower than Forward Indexing and Sequence Pattern Indexing, about 10 to $1,000$ times. The main reason for this is that Prefix-Maximal Indexing consumes a significant amount of time on large $|D'|$ and has no early termination or pruning techniques to alleviate this. As for the Sequence Pattern Indexing, we see the early termination and search space pruning techniques are very effective, with the querying time reduced by 10 to 100 times. If we do not consider Sequence Pattern Indexing, Forward Indexing is really very efficient, and hundreds times faster than Prefix-Maximal Indexing. Our Sequence Pattern Indexing algorithm is the fastest in all the cases, with generally a reduction of a factor of 2 to 5 in the average querying time compared to Forward Indexing. This attributes to the more advanced infrequent sub-phrase filtering (Apriori Principle [4]), together with search space pruning and early termination whose efficiency have been proved by comparing to the same algorithm having those functions turned off, given that Forward Indexing uses only early termination technique. Compared to the very large index size of Forward Indexing, Sequence Pattern Indexing is more preferred given the smallest index size and the smallest average querying time.

5.4 Byte Read Evaluation

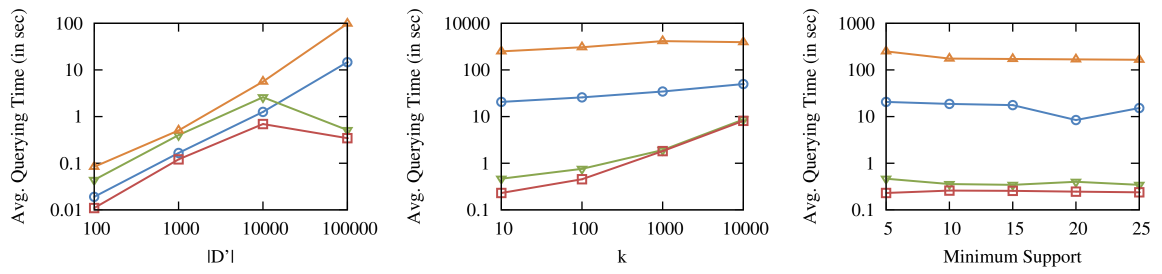
Besides evaluation on average querying time, we further give the detailed evaluation results of byte read on the two datasets – PubMed Titles and PubMed Abstracts – in this section, with varying $|D'|$, varying $min\text{-}sup$, and varying k. The same queries as above are used in the evaluation.

As shown in Figure 9(a) and 9(d), we see that all evaluated algorithms have similar average bytes read with smaller ad-hoc document collection size $|D'|$. However, when $|D'|$ increases to be very large, the two algorithms SeqPattIndex and FowardIndex tend to have significant lower byte read, due to the early termination.



(a) Varying $|D'|$ ($k = 10$, $min\text{-}sup = 5$) (b) Varying k ($min\text{-}sup = 5$) (c) Varying $min\text{-}sup$ ($k = 10$)

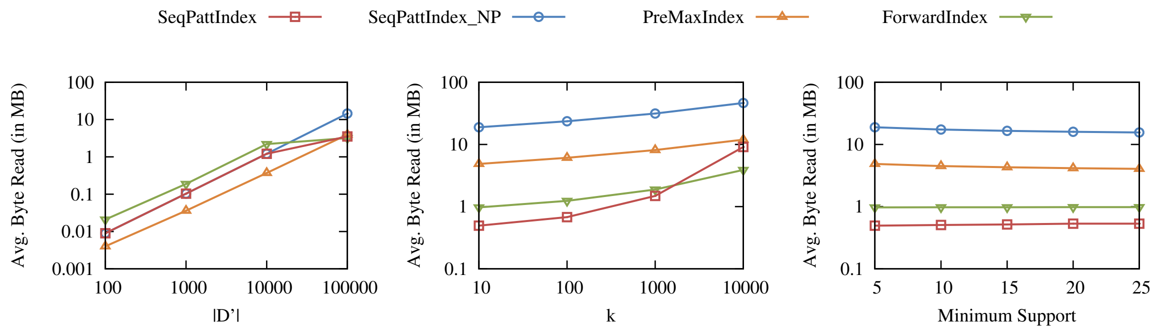
PubMed Titles



(d) Varying $|D'|$ ($k = 10$, $min\text{-}sup = 5$) (e) Varying k ($min\text{-}sup = 5$) (f) Varying $min\text{-}sup$ ($k = 10$)

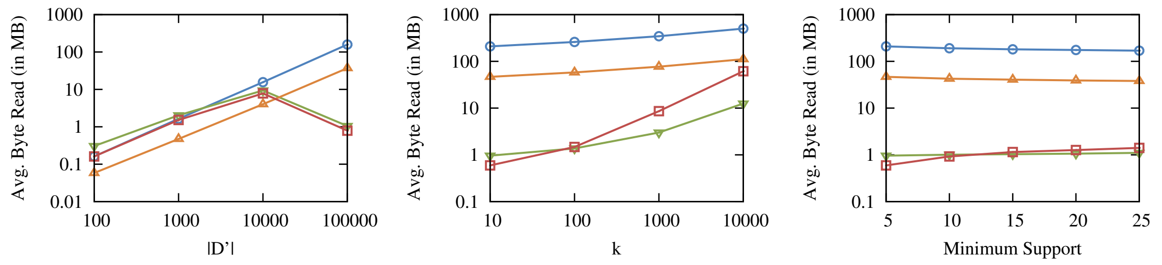
PubMed Abstract

Figure 8: Average Querying Time (in sec)



(a) Varying $|D'|$ ($k = 10$, $min\text{-}sup = 5$) (b) Varying k ($min\text{-}sup = 5$) (c) Varying $min\text{-}sup$ ($k = 10$)

PubMed Titles



(d) Varying $|D'|$ ($k = 10$, $min\text{-}sup = 5$) (e) Varying k ($min\text{-}sup = 5$) (f) Varying $min\text{-}sup$ ($k = 10$)

PubMed Abstracts

Figure 9: Average Byte Read from Index (in MB)

Also, in general, SeqPattIndex and ForwardIndex have similar average bytes read with varying $|D'|$.

Figure 9(b) and 9(e) further provide the results with varying k. Similarly, we can find that SeqPattIndex and ForwardIndex have similar performance due to the existing early termination. While SeqPattIndex has lower byte read than Forward on low k values mainly due to its better search space pruning, its byte read increases fast under large k values, since search space pruning does not work well when k approaches $|D'|$.

For bytes read with varying minimum support threshold $min\text{-}sup$, Figure 9(c) and 9(f) show the results. We can see that SeqPattIndex and ForwardIndex have much better results compared to PreMaxIndex and SeqPattIndex_NP, while all algorithms have nearly the same performance under different $min\text{-}sup$. SeqPattIndex and ForwardIndex have results in the same order in this situation, with SeqPattIndex having the better results, about a factor of 2 to 4 smaller.

In general, on the evaluation of average bytes read, the two algorithms SeqPattIndex and ForwardIndex, which employ early termination, perform significantly better than two others PreMaxIndex and SeqPattIndex_NP. On average, SeqPattIndex has the better byte read performance than ForwardIndex, with about 2 to 10 times fewer bytes read on low k values, while ForwardIndex has better bytes read performance on larger k values.

6. RELATED WORK

The problem of mining top-k interesting phrases has been studied by [15] years ago, but on a static document collection. Until recently, [18] proposes Phrase Inverted Indexing as the first approach to compute the top-k interesting phrases on an ad-hoc document collection. However, as it requires a full scan of the inverted lists of all the documents in the corpus, it runs significantly slow, especially on large-scale datasets. [6] proposes the Forward Indexing approach to solve this problem, storing contained phrase IDs in each document's forward list. Hence, only a scan on the ad-hoc document collection's forward lists is necessary. Together with devised early termination technique, Forward Indexing is significantly faster. Another very different algorithm Prefix-Maximal Indexing is also introduced in [6], focusing on reducing the index size by storing only prefix maximal phrases for each document. However, due to the lack of early termination, Prefix-Maximal Indexing can be thousands times slower than Forward Indexing.

The top-k interesting phrases computation of our algorithm is devised by combining classical $|D'|$-way merge joins and the pattern-growth framework originally used for generalized sequence pattern mining, given that a phrase in our system can be viewed as a special case of a generalized sequence pattern (which allows any length and gaps between items) introduced in [5] at first. The pattern-growth framework was first devised for the PrefixSpan [17] algorithm to mine all frequent sequence patterns, and has been widely adopted by most algorithms developed later. It works by keep extending a prefix with each found locally frequent item, and is in essence a depth-first search algorithm.

Mining top-k interesting patterns has also been studied in the data mining area, often used to find the most discriminative sub-patterns for tasks like classification or clustering. For example, [16] tries to select rules with the top-k highest confidence values on each class from all mined frequent patterns, for item-set datasets classification. In fact, our problem could be solved by adopting the strategy above with the query q of an ad-hoc document collection D' as the class label. However, this would be too costly as for each query hundreds of seconds would be used. Besides, as discussed in Section 5.1, the mining algorithm has to be modified to work on our specialized patterns. Some more advanced data mining algorithms have also been proposed to solve the problem more efficiently and more effectively. [9] proposes to mine the top-k covering rules for each instance directly to help classifying gene expressions, while [20] provides a similar solution for item-set datasets. Recent work [7, 8, 10, 13, 11] are also proposed working on finding most discriminative sub-patterns, with different instance covering technologies, using SVM instead of rule-based classifier, adopting interesting measures like information-gain, using numerical features, and/or classifying uncertain data, etc..

[19] tries to reduce both the size of data cube and querying time on data cube by identifying prefix and suffix structural redundancies, while our system tries to compute top interesting phrases dynamically for each ad-hoc query with both carefully designed index structure storing phrase candidates and efficient algorithm working on the index structure to compute final results, which are in essence two very different problems. [14] works on discovering interesting terms of certain ad-hoc document collections, while [12] tries to find facets from an ad-hoc set of documents. However, identified facets are only about the documents' meta data, unlike our systems working on identifying phrases directly from document texts. Recently, [21] proposes a novel system to try to identify key phrases which could be used in future as queries from an user-inputted text, given an existing text corpus. Phrase candidates are scored by either TF/IDF or mutual information. Wikipedia has also been utilized to help rank phrase candidates.

7. CONCLUSIONS

In this work we have devised a novel indexing technique for mining interesting phrases in ad-hoc document collections. We have coined it Sequence Pattern Indexing (SeqPattIndex) as it harnesses the natural sequential alignment of frequent patterns inside documents. The index created with this approach shows a low level or redundancy compared to existing approaches and can be compressed, as we have shown in this work. We have developed an efficient top-k pattern mining algorithm that operates on the upfront generated indexes and allows for early termination. We have conducted a comprehensive performance study using real world data sets showing that our approach outperforms existing approaches not only in the compactness of the underlying indices but also in the runtime of the top-k mining algorithm.

Acknowledgement

This work has been supported by the Excellence Cluster on Multimodal Computing and Interaction (MMCI).

8. REFERENCES

[1] The dblp computer science bibliography. http://www.informatik.uni-trier.de/ ley/db/.

[2] National science foundation awards. http://www.nsf.gov/awardsearch/.

[3] Pubmed. http://www.ncbi.nlm.nih.gov/pubmed/.

[4] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., 1993. ACM Press.

[5] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society.

[6] S. J. Bedathur, K. Berberich, J. Dittrich, N. Mamoulis, and G. Weikum. Interesting-phrase mining for ad-hoc text analytics. *PVLDB*, 3(1):1348–1357, 2010.

[7] H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *Proceedings of the 23rd International Conference on Data Engineering*, pages 716–725, Istanbul, Turkey, 2007. IEEE.

[8] H. Cheng, X. Yan, J. Han, and P. S. Yu. Direct discriminative pattern mining for effective classification. In *Proceedings of the 24th International Conference on Data Engineering*, pages 169–178, Cancún, México, 2008. IEEE.

[9] G. Cong, K.-L. Tan, A. K. H. Tung, and X. Xu. Mining top-k covering rule groups for gene expression data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 670–681, Baltimore, Maryland, USA, 2005. ACM.

[10] W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. S. Yu, and O. Verscheure. Direct mining of discriminative and essential frequent patterns via model-based search tree. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 230–238, Las Vegas, Nevada, USA, 2008. ACM.

[11] C. Gao and J. Wang. Direct mining of discriminative patterns for classifying uncertain data. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 861–870, Washington, DC, USA, 2010. ACM.

[12] M. A. Hearst. Clustering versus faceted categories for information exploration. *Commun. ACM*, 49(4):59–61, 2006.

[13] H. Kim, S. Kim, T. Weninger, J. Han, and T. F. Abdelzaher. Ndpmine: Efficiently mining discriminative numerical features for pattern-based classification. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010*, pages 35–50, Barcelona, Spain, 2010. Springer.

[14] J. M. Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 91–101, Edmonton, Alberta, Canada, 2002. ACM.

[15] B. Lent, R. Agrawal, and R. Srikant. Discovering trends in text databases. In *KDD*, pages 227–230, 1997.

[16] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the Fourteen ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 80–86, 1998.

[17] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proceedings of the 17th International Conference on Data Engineering*, pages 215–224, Heidelberg, Germany, 2001. IEEE Computer Society.

[18] A. Simitsis, A. Baid, Y. Sismanis, and B. Reinwald. Multidimensional content exploration. *PVLDB*, 1(1):660–671, 2008.

[19] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, and Y. Kotidis. Dwarf: shrinking the petacube. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 464–475, Madison, Wisconsin, 2002. ACM.

[20] J. Wang and G. Karypis. On mining instance-centric classification rules. *IEEE Trans. Knowl. Data Eng.*, 18(11):1497–1511, 2006.

[21] Y. Yang, N. Bansal, W. Dakka, P. G. Ipeirotis, N. Koudas, and D. Papadias. Query by document. In *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009*, pages 34–43, Barcelona, Spain, 2009. ACM.