Phrase Queries with Inverted + Direct Indexes

Kiril Panev and Klaus Berberich

Max Planck Institute for Informatics, Saarbrücken, Germany
{kiril,kberberi}@mpi-inf.mpg.de

Abstract. Phrase queries play an important role in web search and other applications. Traditionally, phrase queries have been processed using a positional inverted index, potentially augmented by selected multi-word sequences (e.g., n-grams or frequent noun phrases). In this work, instead of augmenting the inverted index, we take a radically different approach and leverage the *direct index*, which provides efficient access to compact representations of documents. Modern retrieval systems maintain such a direct index, for instance, to generate snippets or compute proximity features. We present extensions of the established term-at-a-time and document-at-a-time query-processing methods that make effective combined use of the inverted index and the direct index. Our experiments on two real-world document collections using diverse query workloads demonstrate that our methods improve response time substantially without requiring additional index space.

Keywords: information retrieval, search engines, efficiency.

1 Introduction

Identifying documents that literally contain a specific phrase (e.g., "sgt. pepper's lonely hearts club band" or "lucy in the sky with diamonds") is an important task in web search and other applications such as entity-oriented search, information extraction, and plagiarism detection. In web search, for instance, such phrase queries account for up to 8.3% of queries issued by users, as reported by Williams et al. [24]. When not issued explicitly by users, phrase queries can still be issued implicitly by query-segmentation methods [12] or other applications that use search as a service.

Phrase queries are usually processed using a positional inverted index, which keeps for every word from the document collection a posting list recording in which documents and at which offsets therein the word occurs [25]. To process a given phrase query, the posting lists corresponding to the words in the query need to be intersected. This can be done sequentially, processing one word at a time, or in parallel, processing all words at once. Either way, it is typically an expensive operation in practice, since phrase queries tend to contain stopwords. Posting lists corresponding to stopwords, or other frequent words, are very long, resulting in costly data transfer and poor memory locality.

To improve the performance of phrase queries, several methods have been proposed that augment the inverted index by also indexing selected multi-word

B. Benatallah et al. (Eds.): WISE 2014, Part I, LNCS 8786, pp. 156–169, 2014.

sequences in addition to individual words. Those multi-word sequences can be selected, for instance, based on their frequency in a query workload or the document collection, whether they contain a stopword, or other syntactic criteria [21, 24]. While this typically improves performance, since phrase queries can now be processed by intersecting fewer shorter posting lists, it comes at the cost of additional index space and a potential need to select other multi-word sequences as the document collection and the query workload evolve.

One may argue that phrase query processing is just substring matching – a problem investigated intensively by the string processing community. Their solutions such as suffix arrays [16] and permuterm indexes [9]), while increasingly implemented at large scale, are not readily available in today's retrieval systems.

In this work we take a radically different approach. Rather than augmenting the inverted index, we leverage the *direct index* that modern retrieval systems [3, 18] maintain to generate snippets or compute proximity features. The direct index keeps for every document in the collection a compact representation of its contents – typically a list of integer term identifiers. We develop extensions of the established term-at-a-time (TaaT) and document-at-a-time (DaaT) query-processing methods that make effective combined use of the inverted index and the direct index. Our extensions build on a common cost model that captures the trade-off between expensive random accesses (e.g., used to locate a document in the direct index) and inexpensive sequential accesses (e.g., used to read a run of consecutive postings). Taking this trade-off into account, our extended term-at-a-time query processing regularly checks whether verifying the current set of candidates using the direct index is less expensive than reading the remaining posting lists from the inverted index. Likewise, our extended document-at-a-time query processing identifies a set of words upfront, so that intersecting the corresponding posting lists results in a candidate set small enough to be verified using the direct index.

Our extensions are efficient in practice, as demonstrated by our experimental evaluation. Using a corpus from The New York Times and ClueWeb09-B as document collections and query workloads capturing different use cases, we see that our extensions improve response time by a factor of 1.25–2.5 for short queries and 6–80 for long queries. Under the assumption that the direct index is already in place, which is a reasonable one for modern retrieval systems, this improvement comes with no additional index space.

Contributions made in this paper thus include: (i) a novel approach to process phrase queries leveraging the direct index available in modern retrieval systems and requiring no additional index space; (ii) two extensions of established query-processing methods that make effective combined use of the inverted index and the direct index; (iii) comprehensive experiments on two real-world document collections demonstrating the practical viability of our approach.

Organization. In Section 2 we present the technical background of our work. Section 3 then introduces our cost model and our extended query-processing methods. Our experimental evaluation is subject to Section 4. We give an overview of related work in Section 5. Our conclusion and possible future work are presented in Section 6.

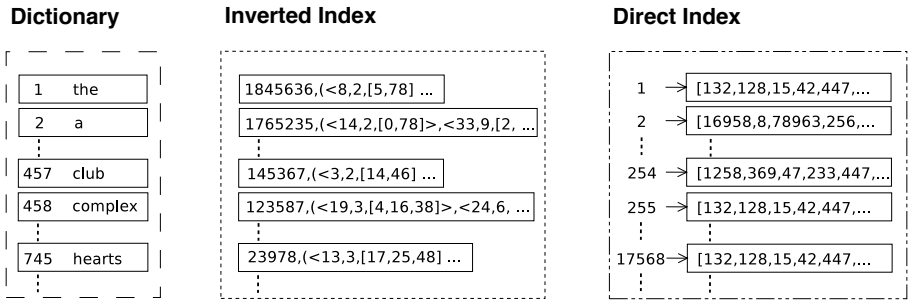


Fig. 1. Prototypic architecture of a modern retrieval system

2 Background

In this section, we introduce our notation and provide some technical background on the approach put forward in this work.

We let $\mathcal{D}$ denote the document collection. Documents therein are sequences of words from a vocabulary $\mathcal{V}$. For a word $v \in \mathcal{V}$ we let $df(v)$ denote its document frequency and $cf(v)$ denote its collection frequency. Likewise, $tf(v, d)$ refers to the term frequency of v in a document $d \in \mathcal{D}$. Phrase queries are also sequences of words from $\mathcal{V}$.

Given a phrase query $\mathbf{q} = \langle q_1, \ldots, q_{|\mathbf{q}|} \rangle$, our objective is to identify the set of all offsets in documents where the given word sequence occurs. Put differently, we only perform a phrase match – aspects of ranking documents based on relevance or proximity are orthogonal to our work.

Figure 1 depicts the prototypic architecture of a modern retrieval system [3, 18], as we assume it in this work, consisting of the following building blocks:

- a *dictionary* that maps words from the vocabulary $\mathcal{V}$ to integer term identifiers, typically implemented using a hash map, a front-coded list, or a B$^+$-tree. By assigning term identifiers in descending order of collection frequency, one can improve the compressibility of the other index structures. Besides term identifiers, the dictionary contains per-word collection statistics such as document frequencies and collection frequencies.
- an *inverted index* that maps term identifiers to posting lists containing information about the occurrences of the corresponding word in the document collection. In a positional inverted index, as used in this work, a posting

$$\langle d, tf(v, d), [o_1, \ldots, o_{tf(v,d)}] \rangle$$

consists of a document identifier d, the term frequency, and an array of offsets where the word v occurs. Note that the term frequency corresponds to the length of the array of offsets and is thus typically represented implicitly. The inverted index is the most important index structure used to match and score documents in response to a query.

– a *direct index* that maps document identifiers to a compact representation of the corresponding document's content. One possible representation is to use arrays of term identifiers, with a special term identifier to mark sentence boundaries. The direct index can be used, for instance, to generate result snippets and compute proximity features.

The concrete implementation of these building blocks depends on whether the index structures are kept in main memory or on secondary storage and whether they reside on a single machine or are spread over a cluster of machines.

Two established methods exist to process phrase queries [5] and other queries that can not profit from optimizations for ranked retrieval such as WAND [4]. Term-at-a-time (TAAT) query processing considers words from the query sequentially in ascending order of their document frequency. While doing so, it maintains a set of candidate documents that can still match the query. When considering a word from the query, the corresponding posting list is intersected with the current set of candidate documents. In the case of phrase queries, for each candidate document the set of offsets is maintained at which the query phrase can still occur. Over time, the set of candidate documents shrinks and query processing can be terminated early once no candidate documents remain. Document-at-a-time query processing (DAAT) considers words from the query in parallel, intersecting their corresponding posting lists. When the same document is seen in all posting lists, meaning that it contains all words from the query, offsets need to be inspected to see whether it contains the query phrase. Using skipping [17] the intersecting of posting lists can be greatly speeded up.

3 Phrase Queries with Inverted + Direct Indexes

In this section, we describe our cost model, characterizing the cost of using the index structures introduced above. Building on it, we develop extensions of established query-processing methods that make combined use of the inverted index and the direct index.

3.1 Cost Model

The cost of random accesses (RA) and sequential accesses (SA) differs substantially across different levels of the memory hierarchy and different storage systems. Thus, as recently reported by Wang et al. [23], on modern magnetic hard disk drives (HDDs) random accesses are 100–130x more expensive than sequential accesses, on modern solid state drives (SSDs) there is still a 2–3x difference, and even in main memory locality of reference matters.

Our cost model abstracts from hardware and implementation details (e.g., block size, cache size, and compression) as well as concrete data structures employed. Its objective is to capture the relative trade-off between random accesses and sequential accesses when using the dictionary, the inverted index, and the direct index. We let c_R denote the cost of a random access and c_S denote the

cost of a sequential access. Our methods developed below are parameterized by the cost ratio c_R/c_S and seek to optimize the total cost

$$c_R \cdot \#R \ + \ c_S \cdot \#S$$

with $\#R$ ($\#S$) denoting the total number of random (sequential) accesses.

We assume the following costs for the index structures described in Section 2. Looking up the term identifier and associated collection statistics in the *dictionary* costs a single random access per word v from the query. Reading the entire posting list associated with word v from the *inverted index* incurs a single random access, to locate the posting list, and $df(v)$ sequential accesses to read all postings therein. On a HDD, for instance, assuming that the posting list is stored contiguously, this corresponds to seeking to its start followed by sequentially reading its (possibly compressed) contents. Finally, retrieving the compact representation of a document d from the *direct index* amounts to a single random access. We hence disregard (differences in) document lengths and assume that compact document representations are small enough to be fetched in a single operation.

3.2 Term-at-a-Time

Term-at-a-time query processing considers words from the query sequentially in ascending order of their document frequency. Our example phrase query "lucy in the sky with diamonds" from the introduction would thus be processed by considering its words in the order $\langle$ lucy, diamonds, sky, with, in, the $\rangle$. After having processed the first two words lucy and diamonds, the set of candidates consists of only those documents that contain both words at exactly the right distance. If this set is small enough, instead of reading the remaining posting lists, it can be beneficial to switch over to the direct index and verify which candidates contain the not-yet-considered words at the right offsets.

Let $\langle\, v_1, v_2, \ldots, v_n \,\rangle$ denote the sequence of distinct words from the phrase query $\mathbf{q}$ in ascending order of their document frequency (i.e., $df(v_i) \leq df(v_{i+1})$). After the first $k > 1$ words have been processed, we can compare the cost of continuing to use the inverted index against verifying the set of candidates using the direct index. More precisely, according to our cost model, the cost of continuing to use the inverted index is

$$c_R \cdot (n-k) \ + \ c_S \cdot \sum_{i=k+1}^{n} df(v_i) \ ,$$

for accessing and reading the remaining $(n-k)$ posting lists. The cost of verifying all documents from the current candidate set $\mathcal{C}_k$ is

$$c_R \cdot |\mathcal{C}_k| \ .$$

Our extension of TAAT determines the two costs whenever a posting list has been completely read and switches over to verifying the remaining candidate documents using the direct index once this is beneficial.

3.3 Document-at-a-Time

Document-at-a-time query processing considers words from the query in parallel and does not (need to) maintain a set of candidate documents. This has two ramifications for our extension. First, we need to decide upfront which words from the query to process with the inverted index. Second, to do so, we have to rely on an estimate of the number of candidate documents left after processing a set of words from the query using the inverted index.

To make this more tangible, consider again our example phrase query "lucy in the sky with diamonds". We could, for instance, use the inverted index to process the words in { lucy, diamonds } and verify the remaining candidate documents using the direct index. The cost for the former can be determined exactly. The cost for the latter depends on how many documents contain the two words at the right distance. Upfront, however, this number can only be estimated.

Let $\langle\, v_1, v_2, \ldots, v_n \,\rangle$ denote the sequence of distinct words from the phrase query $\mathbf{q}$, again in ascending order of their document frequency. Finding the cost-optimal subset I of words from the query to be processed using the inverted index can be cast into the following optimization problem

$$\underset{I \subseteq \langle\, v_1, v_2, \ldots, v_n \,\rangle}{\arg\min} \; c_R \cdot |I| \;+\; c_S \cdot \sum_{v \in I} df(v) \;+\; c_R \cdot |\hat{\mathcal{C}}_I| \quad \text{s.t.} \quad I \neq \emptyset \,.$$

The first two summands capture the cost of processing the words from I using the inverted index; the last summand captures the estimated cost of verifying candidate documents using the direct index. We demand that at least one word is processed using the inverted index (i.e., $I \neq \emptyset$). Otherwise, all documents from the collection would have to be matched against the query phrase using the direct index, which is possible in principle but unlikely a preferable option. In the above formula, $|\hat{\mathcal{C}}_I|$ denotes an estimate of how many candidate documents are left after words from I have been processed using the inverted index.

Assuming that no additional collection statistics (e.g., about n-grams) are available, we estimate this cardinality as

$$|\hat{\mathcal{C}}_I| = |\mathcal{D}| \cdot \prod_{v \in I} \frac{df(v)}{|\mathcal{D}|} \,,$$

thus making an independence assumption about word occurrences in documents. More elaborate cardinality estimations, taking into account the order of words and their dependencies, would require additional statistics (e.g., about n-grams) consuming additional space.

It turns out that an optimal solution to our optimization problem can be determined efficiently. To this end, observe that $\{v_1, \ldots, v_k\}$ has minimal cost among all k-subsets of words from the query. This follows from a simple exchange argument: We can replace words in any k-subset by words from $\{v_1, \ldots, v_k\}$ having at most the same document frequency without increasing the overall cost. It is thus sufficient to determine the $\{v_1, \ldots, v_k\}$, consisting of the k rarest words from the query, that minimizes the overall cost. This can be done in time $\mathcal{O}(|\mathbf{q}| \log |\mathbf{q}|)$

– dominated by the cost of sorting words from the query.

Both our extensions ignore the cost for looking up in the dictionary information about the words from the query. This is a fixed cost which all query-processing methods have to pay. Further, when verifying a candidate document using the direct index, our extensions do not need to employ a string matching algorithm such as Knuth-Morris-Pratt [15], since they already know candidate offsets where the phrase can still occur.

4 Experimental Evaluation

We now present experiments evaluating the approach put forward in this work.

4.1 Setup and Datasets

Document Collections. We use two publicly available real-world document collections for our experiments: (i) The New York Times Annotated Corpus (NYT) [2], which consists of more than 1.8 million newspapers articles from the period 1987-2007 and (ii) ClueWeb09-B (CW) [1] as a collection of about 50 million English web documents crawled in 2009. Table 1 reports detailed characteristics of the two document collections.

Table 1. Document collection characteristics

	NYT	**CW**
# Documents	1,831,109	50,221,915
# Term occurrences	1,113,542,501	23,208,133,648
# Terms	1,833,817	51,322,342
# Sentences	49,622,213	1,112,364,572
Size of inverted index (GBytes)	2.98	60.21
Size of direct index (GBytes)	2.22	48.65

Workloads. To cover diverse use cases, we use four different workloads per document collection: (i) a query log from the MSN search engine released in 2006 (MSN), (ii) a subset of the aforementioned query log that only contains explicit phrase queries (i.e., those put into quotes by users) (MSNP), (iii) entity labels as captured in the `rdfs:label` relation of the YAGO2 knowledge base [14] (YAGO), (iv) a randomly selected subset of one million sentences from the respective document collection (NYTS/CWS). While the first two workloads capture web search as a use case, the YAGO workload mimics entity-oriented search, assuming that entities are retrieved based on their known labels. The NYTS/CWS workloads capture plagiarism detection as a use case, assuming

that exact replicas of text fragments need to be retrieved. Queries in the first three workloads are short consisting of about three words on average; queries in the sentence-based workloads are –by design– longer, consisting of about twenty words on average. For each workload, we consider a randomly selected subset of $7,500$ distinct queries. Table 2 reports more detailed characteristics of the considered workloads.

Table 2. Workload characteristics

	# Queries	# Distinct Queries	ø Query Length
YAGO	5,720,063	4,599,745	2.45
MSN	10,428,651	5,627,838	3.58
MSNP	131,857	105,825	3.37
NYTS	1,000,000	970,051	21.66
CWS	1,000,000	929,607	20.55

Methods under comparison are the following: (i) TAAT-I and (ii) DAAT-I as the established query-processing methods that solely rely on the inverted index, (iii) TAAT-I+D and (iv) DAAT-I+D as our extensions that also use the direct index. The latter two are parameterized by the cost ratio c_R/c_S, which we choose from $\{10, 100, 1000, 10{,}000\}$ and report with the method name. Thus, when we refer to TAAT-I+D(100), the cost ratio is set as $c_R/c_S = 100$.

Implementation. All our methods have been implemented in Java. Experiments are conducted on a a server-class machine having 512 GB of main memory, four Intel Xeon E7-4870 10-core CPUs, running Debian GNU/Linux as an operating system, and using Oracle JVM 1.7.0_25. The direct index keeps for every document from the collection an array of integer term identifiers. The inverted index keeps for every word a posting list organized as three aligned arrays containing document identifiers (in ascending order), term frequencies, and offsets. With this implementation, skipping in posting lists is implemented using galloping search. When determining response times of a method for a specific query, we execute the query once to warm caches, and determine the average response time based on three subsequent executions. Here, response time is the time that passes between receiving the query and returning the set of offsets where the phrase occurs in the document collection. To shield us from distortion due to garbage-collection pauses, we use the concurrent mark-sweep garbage collector.

4.2 Experimental Results

We compare TAAT-I+D and DAAT-I+D against their respective baseline TAAT-I and DAAT-I in terms of query response times. To determine the response time of a method for a specific query, we execute it once to warm caches,

and determine the average response time based on three subsequent executions. Here, all 7,500 queries from each workload are taken into account, including those that do not return results. Table 3 reports average response times in milliseconds. Improvements of our extensions over their respective baseline reported therein are statistically significant ($p < 0.01$), as measured by a paired Student's t-test.

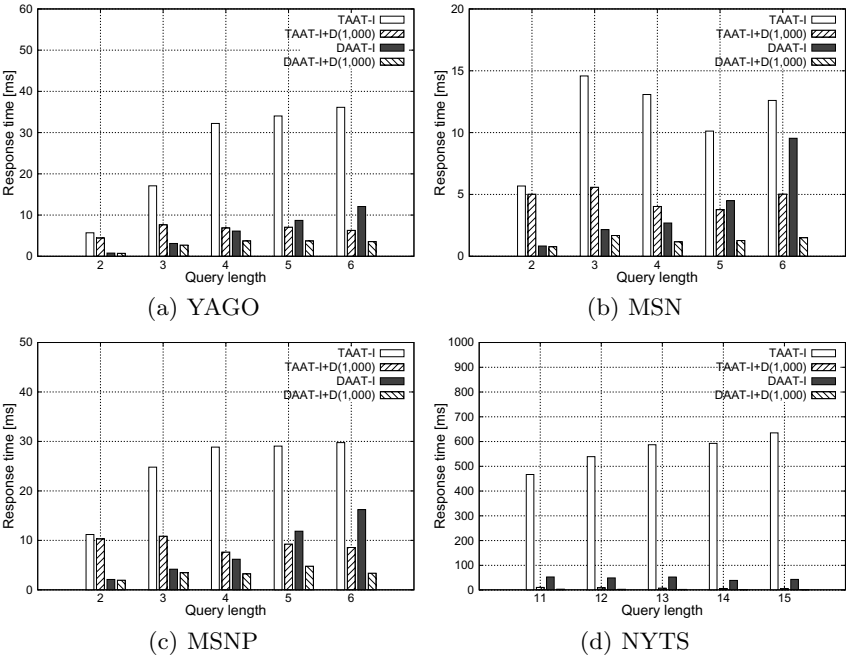
We observe that TAAT-I+D and DAAT-I+D perform significantly better than their respective baseline. For $c_R/c_S = 1,000$, as a concrete setting, our extensions improve response time by 1.25–2.5× for the workloads consisting of relatively short queries. For the verbose workloads consisting of sentences from the respective document collection, our extensions improve response time by 6–80×. Across both document collections we see slightly smaller improvements for DAAT-I+D. This is expected given that DAAT-I is the stronger among the baselines and DAAT-I+D has to decide upfront which words to process with which index. The performance of DAAT-I+D thus depends critically on the accuracy of cardinality estimation.

To see when our extensions are particularly effective, consider the phrase query "the great library of alexandria" from the YAGO workload. On the NYT document collection, the response time for this query drops from 444.7 *ms* with TAAT-I to only 5.3 *ms* with TAAT-I+D(1,000) and from 12.3 *ms* with DAAT-I to 1.3 *ms* with DAAT-I+D(1,000). Both TAAT-I and DAAT-I read the entire posting lists for all words from the query, resulting in a total of 3,743,120 sequential accesses. With TAAT-I+D(1,000), though, after processing the posting lists for alexandria and library, thereby making only 43,148 sequential accesses, we are left with only 65 candidates, which can quickly be verified using the forward index. Based on its cardinality estimate, DAAT-I+D(1,000) also selects the set { alexandria, library }, resulting in the same cost. However, it is difficult for our extensions to improve response times of short queries whose words have similar document frequencies. In these cases, TAAT-I+D(1,000) and DAAT-I+D(1,000) fall back onto using only the inverted index, since it is not beneficial, according to our cost model, to switch over to the forward index. For the phrase query "sky eye" on the NYT document collection, we have document frequencies $df(\text{sky}) = 35{,}798$ and $df(\text{eye}) = 80{,}830$. After processing the term sky, only 35,798 candidates are left, so that according to our cost model it is better to process the term eye using the inverted index. This results in 80,830 sequential accesses, which is substantially less expensive than performing the 35,798 random accesses required to verify candidates.

We also investigate where our gains come from and whether our extensions perform particularly well for shorter or longer queries. Figure 2 and Figure 3 show average response times by query length for NYT and CW as observed on the different workloads. Comparing the two baselines, we observe that query response time tends to increases with query length for both, but stagnates at some point for TAAT-I. This is natural since TAAT-I can often terminate query processing early once no candidates are left, whereas DAAT-I has to continue reading all posting lists. On the sentence-based workloads, in which every query

Table 3. Query response time (in ms)

	NYT				CW			
	YAGO	MSN	MSNP	NYTS	YAGO	MSN	MSNP	CWS
TAAT-I	14.13	10.94	21.87	810.01	1,767.02	787.84	2,428.85	16,962.03
TAAT-I+D(10)	4.64	4.47	9.26	12.95	1,498.60	1,948.41	3,649.31	4,067.63
TAAT-I+D(100)	4.51	4.11	8.56	10.76	860.68	915.58	2,048.24	3,161.38
TAAT-I+D(1,000)	5.76	4.77	9.89	10.10	676.23	497.94	1,557.80	1,919.78
TAAT-I+D(10,000)	7.32	5.73	11.95	11.47	699.19	480.97	1,389.16	1,500.35
DAAT-I	2.85	3.22	6.62	38.47	55.52	72.92	138.20	446.78
DAAT-I+D(10)	1.24	1.03	2.05	2.93	194.07	268.93	470.53	297.02
DAAT-I+D(100)	1.43	0.97	2.38	3.60	51.33	57.23	95.48	68.52
DAAT-I+D(1,000)	1.82	1.25	3.24	4.77	40.22	35.24	79.16	71.17
DAAT-I+D(10,000)	2.13	1.60	4.03	6.56	44.55	37.08	94.01	87.71



(a) YAGO (b) MSN

(c) MSNP (d) NYTS

Fig. 2. Query response time by query length (in ms) on NYT

has by design at least one result, query response times of TAAT-I keep increasing with query length. On both document collections and all workloads, our extensions improve query response time consistently across different query lengths. Only when comparing DAAT-I and DAAT-I+D we observe a widening gap. As explained before, query response times increase with query length for DAAT-I, whereas our extension DAAT-I+D manages to keep them relatively constant.

Summary

Our extensions TAAT-I+D and DAAT-I+D significantly improve query response time across different query lengths, as demonstrated by our experiments on two document collections and four different workloads. It is worth emphasizing once more that our extensions solely rely on the inverted index and direct index, which are available in modern retrieval system anyway, so that their improvement comes with no additional index space.

5 Related Work

The use of phrases dates back to the early days of Information Retrieval. Since indexing all possible phrases is prohibitively expensive, researchers initially focused on identifying salient phrases in documents to index. Salton et al. [19],

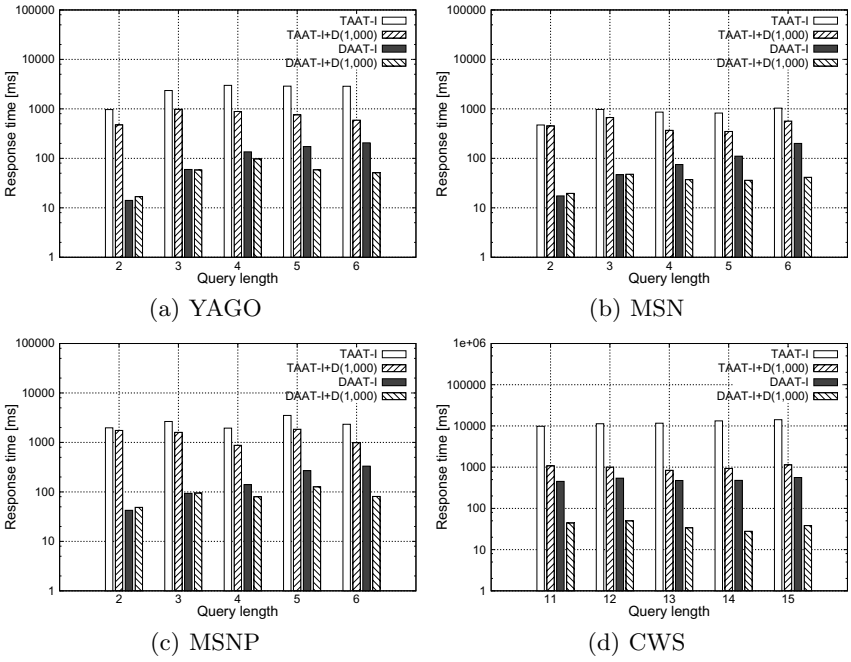


Fig. 3. Query response time by query length (in ms) on CW

Fagan [8], and Gutwin et al. [11] develop methods to this end based on statistical and syntactical analysis. With those methods, one could thus look up documents that contain one of the selected salient phrases.

To support arbitrary phrase queries, more recent research has looked into systematically augmenting the inverted index by selected multi-word sequences in addition to individual words. Williams et al. [24] develop the *nextword index*, which indexes bigrams containing a stopword in addition to individual words. They also discuss how to make effective combined use of the nextword index, a partial phrase index, which caches the phrases most often occurring in the query workload, and a standard inverted index. Chang and Po [6] describe an approach to select variable-length multi-word sequences taking into account part-of-speech information, frequency in the document collection, and frequency in a query workload. Transier and Sanders [21] present a method that works "out of the box" without any knowledge about the query workload. Their approach selects bigrams that are beneficial to keep because intersecting the posting lists of their constituting words is expensive. One shortcoming of all of these approaches is that they require additional index space. Our method, in contrast, leverages the direct index, which is anyway available in modern retrieval systems. Interestingly, Transier and Sanders [21] describe a somewhat similar approach that uses a non-positional inverted index in combination with the direct direct index. Issuing a conjunctive Boolean query against the inverted index they identify

candidate documents. Those are then fetched from the direct direct index and matched against the phrase query using the Knuth-Morris-Pratt algorithm [15]. Our approach, in contrast, foregoes reading all posting lists and verifies candidates by probing specific offsets as opposed to performing a phrase match.

Other related yet orthogonal work has looked into how positional inverted indexes can be improved for specific use cases. Shan et al [20] propose a *flat position index* for phrase querying. It views the whole document collection as a single word sequence, so that posting lists consist of offsets only but do not contain any document identifiers. Only when an occurrence of the query phrase is found, the corresponding document identifier is reconstructed from a separate index. He and Suel [13] develop a positional inverted index for versioned document collections (e.g., web archives), exploiting the typically large overlap between consecutive versions of the same document.

It is worth mentioning that, at its core, phrase querying is a substring matching problem, which has been studied intensively by the string processing community. State-of-the-art solutions such as suffix arrays [16] and permuterm indexes [9] were originally designed to work in main memory. Scaling such data structures to large-scale document collections as we consider them is an exciting ongoing direction of research with good progress [7, 10, 22] in the recent past.

6 Conclusion

We have proposed extensions of the established term-at-a-time and document-at-a-time query processing methods that make combined use of the inverted index and the direct index. Given that modern retrieval systems keep a direct index for snippet generation and computing proximity features, our extensions do not require additional index space. Our experiments show that these extensions outperform their respective baseline when the cost ratio parameter is set appropriately. We observed substantial improvements in response time in particular for verbose phrase queries consisting of many words. We believe that this makes our approach relevant for applications, such as entity-oriented search or plagiarism detection, that rely on machine-generated verbose phrase queries.

As part of our future research, we plan to (i) investigate more elaborate cardinality estimation techniques and their effect on query-processing performance, (ii) refine our cost model to take into account the effect of skipping and other optimizations, (iii) adapt our approach to also handle more expressive queries (e.g., including wildcards and/or proximity constraints).

References

[1] The ClueWeb09 Dataset, `http://lemurproject.org/clueweb09/`
[2] The New York Times Annotated Corpus, `http://corpus.nytimes.com/`
[3] Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer Networks 30(1-7), 107–117 (1998)
[4] Broder, A.Z., Carmel, D., Herscovici, M., Soffer, A., Zien, J.Y.: Efficient query evaluation using a two-level retrieval process. In: CIKM 2003 (2003)

[5] Büttcher, S., Clarke, C., Cormack, G.V.: Information Retrieval: Implementing and Evaluating Search Engines. MIT Press (2010)
[6] Chang, M., Poon, C.K.: Efficient phrase querying with common phrase index. Inf. Process. Manage. 44(2), 756–769 (2008)
[7] Culpepper, J.S., Petri, M., Scholer, F.: Efficient in-memory top-k document retrieval. In: SIGIR 2012 (2012)
[8] Fagan, J.: Automatic phrase indexing for document retrieval. In: SIGIR 1987 (1987)
[9] Ferragina, P., Venturini, R.: The compressed permuterm index. ACM Transactions on Algorithms 7(1) (2010)
[10] Gog, S., Moffat, A., Culpepper, J.S., Turpin, A., Wirth, A.: Large-scale pattern search using reduced-space on-disk suffix arrays. CoRR abs/1303.6481 (2013)
[11] Gutwin, C., Paynter, G., Witten, I., Nevill-Manning, C., Frank, E.: Improving browsing in digital libraries with keyphrase indexes. Decision Support Systems 27(1-2), 81–104 (1999)
[12] Hagen, M., Potthast, M., Beyer, A., Stein, B.: Towards optimum query segmentation: in doubt without. In: CIKM 2012 (2012)
[13] He, J., Suel, T.: Optimizing positional index structures for versioned document collections. In: SIGIR 2012 (2012)
[14] Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: Yago2: A spatially and temporally enhanced knowledge base from wikipedia. Artif. Intell. 194, 28–61 (2013)
[15] Knuth, D., Morris, J. J., Pratt, V.: Fast pattern matching in strings. SIAM Journal on Computing 6(2), 323–350 (1977)
[16] Manber, U., Myers, E.W.: Suffix arrays: A new method for on-line string searches. SIAM J. Comput. 22(5), 935–948 (1993)
[17] Moffat, A., Zobel, J.: Self-indexing inverted files for fast text retrieval. ACM Trans. Inf. Syst. 14(4), 349–379 (1996)
[18] Ounis, I., Amati, G., Plachouras, V., He, B., Macdonald, C., Johnson, D.: Terrier information retrieval platform. In: Losada, D.E., Fernández-Luna, J.M. (eds.) ECIR 2005. LNCS, vol. 3408, pp. 517–519. Springer, Heidelberg (2005)
[19] Salton, G., Yang, C.S., Yu, C.T.: A theory of term importance in automatic text analysis. Journal of the American Society for Information Science 26(1), 33–44 (1975)
[20] Shan, D., Zhao, W.X., He, J., Yan, R., Yan, H., Li, X.: Efficient phrase querying with flat position index. In: CIKM 2011 (2011)
[21] Transier, F., Sanders, P.: Out of the box phrase indexing. In: Amir, A., Turpin, A., Moffat, A. (eds.) SPIRE 2008. LNCS, vol. 5280, pp. 200–211. Springer, Heidelberg (2008)
[22] Vigna, S.: Quasi-succinct indices. In: WSDM 2013 (2013)
[23] Wang, J., Lo, E., Yiu, M.L., Tong, J., Wang, G., Liu, X.: The impact of solid state drive on search engine cache management. In: SIGIR 2013 (2013)
[24] Williams, H.E., Zobel, J., Bahle, D.: Fast phrase querying with combined indexes. ACM Trans. Inf. Syst. 22(4), 573–594 (2004)
[25] Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Comput. Surv. 38(2) (2006)