A Method for Approximating Document Frequency in Top-k Document Retrieval

Tokinori Suzuki[1,a)] Atsushi Fujii[1]

Abstract:
Top-k document retrieval is essential task for real world applications such as web search and data mining. A new class of indexes derived from *suffix array* family have been studied in this decade. This indexes are expected to improve efficiency of document retrieval task and support *general document retrieval*, where documents are not written in natural languages only, on its algorithmic backgrounds. One of main feature of the indexes is indexing all of substrings in document collection. The indexes ,therefore, have difficulty on handling *document frequency* in terms of space. Previous work [3] provided the circumevent method to weight pesudo terms, which is not related to *documente frequcency* purely. We propsed two methods: to approximate *document frequency* of terms from the indexes strucuture and to use *term count* for term wightening. Our main contribution is providing simple methods that can run on undorned indexes. Experimental results show that our methods are great on efficiency and effectiveness trade-off in practical document retrieval task.

Keywords: Experimentation, Performance, Text Indexing, String Processing

1. Introduction

Reflecting the growing volume of textual information mainly available via the World Wide Web, effective and efficient document retrieval has of late become crucial. While it is prohibitive to exhaustively search the Web for the documents that are potentially related to a user query, it is usually important to find a limited number of highly related documents with a quite minimal computation cost. This task, that we dubbed *cheating top-k document retrieval*, is continuously one of the main topics in information retrieval, data engineering, and string processing communities.

Whereas in principle *top-k document retrieval* is the process to precisely find the k highest-scoring documents for a query, in practice the process is terminated after finding a predetermined number of documents that are likely to be among the highest-scoring documents, so that the computation cost can substantially be reduced while maintaining the retrieval accuracy. This technique, termed "early termination", involves organization and traversal for the underlying index. From an index structure point of view, existing methods for *cheating top-k document retrieval* can be divided into two categories.

The first category relies on an inverted index, which has been dominant index structure in document retrieval for a long time. Methods based on heuristics for *cheating top-k document retrieval* have been well investigated. With *max-score* ,for example, *posting lists* are arranged in high *term frequency* order. On the traversing, we can maintain score to avoid accessing irrelevant fragments in the index [5]. A major drawback of the inverted index is that index terms must be predetermined prior to the indexing process and thus terms not indexed cannot be used for querying purposes.

The second category relies on a full-text indexes derived from *suffix array*, such as *FM-index*, which can alleviate the above mentioned problem related to the inverted index. A number of methods have recently been proposed to improve efficiency [10,11] and effectiveness [2–4]. One of the state-of-the-art framework in this category, which has been proposed by Culpepper et al. [2], uses a *FM-index structured* and *document array* represented by *wavelet tree (WT)*, and traverse *document array* in a greedy fashion to find documents in which a query term appears with a high *term frequency (tf)*. An advantage of this method is that *tf* can be calculated while searching for documents. However, this method cannot handle *document frequency (df)*, which is usually more effective than *tf* in retrieval accuracy.

Motivated by the above discussion, in this paper we propose methods to approximate *df* values in the WT-based *FM-index*. In other words, we intend to improve the effectiveness of the method proposed by Culpepper et al.. while maintaining its efficiency. We also compare the effectiveness of different methods for the early termination experimentally.

In Section 2 we discusses other research related to *top-k document retrieval*. We describes our methods in section 3 and evaluates their effectiveness experimentally in section 4.

2. Related work

Gagie et al. [6] presented range quantile queries on wavelet tree to *document listing problem*. As a document array D is represented by wavelet tree and given range D[*sp, ep*], their query can get possible distinct documents within the range in order.

[1] Tokyo Institute of Technology, Meguro, Tokyo 152–8552, Japan
[a)] tokinori.suzuki@gmail.com

The query descends to next level nodes(left and right), calculating next node interval(*sp' and ep'*) over *rank* operation. In order to find successive document id, the traversal prefer to go down left node anytime the interval D[*sp', ep'*] is not empty, otherwise it goes to right node. At the end of traversal, i-th and lower documents and the those interval on leaf corresponding to *tf(p,d)* are gathered.

Culpepper et al. [2] extended above queries for efficiency on application of Top-k document retrieval. Their main contribution was the observation that the interval on queries reflects *tf(p,d)*. The larger interval nodes of wavelet tree possibly contain the documents with high *tf(p,d)*. Their method ,therefore ,prioritizes traversal of wavelet tree by the size of nodes interval D[*sp', ep'*]. The traversal prefers to descend to the node with lager interval on same level(**GREEDY** fashion), maintaining the priority using priority queue of (node, interval). When it reaches leaf node, it is possible to report document id and its tf in high tf order. Their method works well on search task where query consists of one term, because there are no needs to care about weights among query terms. However, scoring using only tf is on limited scenario on document retrieval.

Navarro and Valenzuela [11] achieved improving further efficiency of search with precomputed Top-k answer. They sample $k^*(\geq k)$ most frequent documents within the range on D[*sp', ep'*] in advance, where $ep' - sp' = parameter\ g$. They store the result to additional data structure(derived from HSV concept [7]) using suffix sampling. When range(*sp and ep*) corresponding to query comes, it starts with the precomputed results and updates it considering sub-intervals [*sp, sp'-1*] and [*ep'+1, ep*]. While they applied *HSV* auxiliary structure to their index structures, this search task still stayed in sub-problem of document retrieval(scoring with *tf* only). Culpepper et al. [3] employed the auxiliary structure in generating pseudo terms in whole documents collection. [1]

3. Proposed methods

We describe our two method to estimate *Document Frequency(df)* to index structure consisted of *FM-Index* and document array with *wavelet tree*. It is hard to handle df on search time, because of retrieval target domain; We cannot define query terms(substrings) in advance, and it is not realistic to store *df* of all substrings of document collection in terms of space. The work [3], therefore, provided the way to circumvent this problem with *impact ordering*.

Our approach is to estimate *df* on the index([2,4]), and do not require any other auxiliary data structures. Figure 1 shows our system: *FM-Index over T and document array*. *Document array* store document ids that correspond every suffix of T. Both of components are represented by wavelet tree. Drawing *suffix array and suffixes* for illustration, we do not store the structures.

3.1 GREEDY traversal with estimating Document Frequency

The idea of first solution is to utilize the tree structure of *wavelet tree of document array*. Recall that we know the height of *wavelet tree*, that is $\log d$. This information can enhance the Culpepper's **GREEDY** [2] traversal. Their method gather documents in high $tf(p,d)$ order, prioritizing the nodes to descend with the interval *sp and ep*. The traversal can eliminate redundant part of document collection, where the documents with p are less significant(low tf).

SA	Suffix	BWT	D	
13	$	B	3	
6	ABBAB$	A	1	
2	ABAABBAB	A	0	
9	BAB	B	2	
5	AABBAB$	B	1	⇐ sp
1	AABAABBAB	A	0	
0	AAABAABBAB	$	0	
11	AB$	B	3	
7	ABBAB	$	2	
3	ABAABBAB$	$	1	⇐ ep
12	B$	A	3	
8	BBAB	A	2	
4	BAABBAB$	A	1	
10	BAB$	$	3	

Fig. 1 T="AAABAABBAB" is the concatenation of four documents delimited by terminal symbol $. The last two columns are index over T:BWT is T^{BWT} as a *FM-Index* and D is document array. BWT supports to define the range *sp and ep* along with pattern p. Given pattern "A", for example, it returns 4 and 9. D represented by *wavelettree* supports to calculate the statistic within the range

First approach is approximating occurrences of documents hanging down from nodes pruned by the traversal. Algorithm 1 shows the perspective of our method. As the traversal favors bigger range node in greedy fashion from root to leaf, it encounters the node $|ep' - sp'| < \sigma$ to prune. Some documents may occur within the range repeatedly. It is obvious that the number of individual documents under the node is at most $2^{\log d - l}$ where l is level of the node, because *wavelet tree* is complete binary tree. Choosing a smaller value from $2^{\log d - l}$ and $ep' - ep'$ as approximate occurrence of documents, we update $approximate\ df(kDF)$(line 16 - 20 and 25 - 29 in Algorithm 1). kDF value is also added one when the candidate document is found on leaf node.

As the result of our traversal, kDF can be formulated as follows;

$$kDF(p) = k + \begin{cases} \sum_{node} 2^{depth} & ((ep' - sp') > 2^{depth}) \\ \sum_{node}(ep' - sp') & (otherwise) \end{cases} \quad (1)$$

Where depth is the level of the pruned node enumerating from leaves, *sp'* and *ep'* are allocated range of the node.

3.2 Term count for approximating document frequency

The second solution is to use *term count* instead of *document frequency*. *term count* is the number of occurrence of terms counted in the document collection duplicately. It is well studied that *document frequency* and *term count* are strong correlated each other in practical data set ,such as web page [8,9]. We substituted *term count* for approximate value of *df*, since it corresponds to the range *ep - sp*, where all of suffixes sharing same pattern are gathered, in our index. Thus, we can obtain the value as the equation 2. This is our observation that *ranges* in this index are quite

Algorithm 1 GREEDY with DF estimation

Require: *sp and ep along with query term and k*
Ensure: *A list of k candidate documents*
1: *A heap H, H.PUSH*$(l, [sp, ep])$, *A priority queue PQ*, $PQ \leftarrow \{\}$
2: *An approximate DF kDF*, $kDF \leftarrow 0$
3: $l \leftarrow$ *root of Doc Array*, *# of candidate c*, $c \leftarrow 0$
4: **while** $c < k$ *and* $H \neq \{\}$ **do**
5: $l, [sp', ep'] \leftarrow H.POP()$
6: **if** l *is leaf* **then**
7: $PQ.ENQUEUE(l.docid,\ ep' - sp' + 1)$
8: $c \leftarrow c + 1$
9: $kDF \leftarrow kDF\ + 1$
10: **else**
11: $[s_0, e_0] \leftarrow [B_l.rank_0(sp'), B_l.rank_0(ep')]$
12: $[s_1, e_1] \leftarrow [B_l.rank_1(sp'), B_l.rank_1(ep')]$
13: **if** $e_1 - s_1 > \sigma$ **then**
14: $H.PUSH(l.left, [sp_0, ep_0])$
15: **else**
16: **if** $[sp_0, ep_0] < 2^{docarray.depth\ -\ l}$ **then**
17: $kDF \leftarrow kDF\ + [sp_0, ep_0]$
18: **else**
19: $kDF \leftarrow kDF\ + 2^{docarray.depth\ -\ l}$
20: **end if**
21: **end if**
22: **if** $e_1 - s_1 > \sigma$ **then**
23: $H.PUSH(l.right, [sp_1, ep_1])$
24: **else**
25: **if** $[sp_1, ep_1] < 2^{docarray.depth\ -\ l}$ **then**
26: $kDF \leftarrow kDF\ + [sp_1, ep_1]$
27: **else**
28: $kDF \leftarrow kDF\ + 2^{docarray.depth\ -\ l}$
29: **end if**
30: **end if**
31: **end if**
32: **end while**
33: *return PQ and kDF*

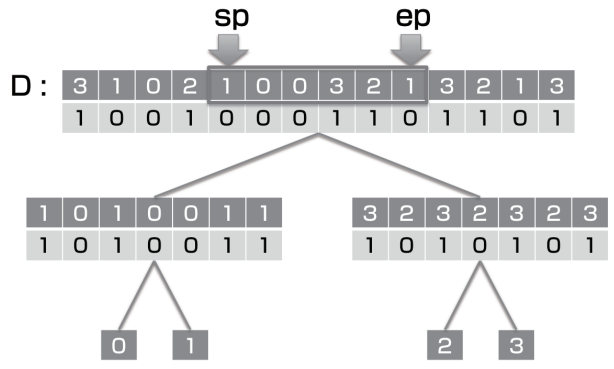

Fig. 2 Document Array D represented by *wavelet tree* for $D = \{3, 1, 0, 2, 1, 0, 0, 3, 2, 1, 3, 2, 1, 3\}$. The box shows the range corresponding to pattern p in SA.

term count, not by previous works. This approximation have the advantages; it is easy to get approximate value, because of *range* and we can get the values before start traversing *document array*. Therefore, the second advantage also can prioritize query terms to process in indexes.

$$kDF(p) = ep - sp \tag{2}$$

4. Experiments

We conducted two experiments to evaluate our methods on efficiency and effectiveness of practical search task using NTCIR-2 test collection. To evaluate methods, we measured time to process queries as efficiency measures and effectiveness measures such as mean average precision using the topics. We illustrated the efficiency and effectiveness trade-offs of methods for comparison at the end of this experiments.

4.1 Test Collection

We conducted our experiments with NTCIR-1 and 2 monolingual IR task, E-E task and E-collection [*1]. E-E task is the task that search English documents using topics also written in English. E-collection is composed of texts extracted from title, authors' name and abstract on scientific paper in English. The number of documents is 322,058, and total size of the collection is 438MB. A sum of 49 topics(topic number e0101-0149) and associated relevance judgments were prepared in E-E task. Relevance assessment had carried out in four grades: "highly relevant (S)", "relevant (A)", "partially relevant (B)" and "non-relevant (C)" for each document. We used "highly relevant (S)" and "relevant (A)" as relevant documents(Level 1) to calculate effectiveness measures.

Our framework and algorithms were implemented with C/C++(gcc 4.2.1, -O3 option). All of experiments were run on a system with 2.3Ghz Intel Core i5 Processor(2-Core) with 8GB RAM.

4.2 Query setting

We used two manners to generate queries in a series of experiments: *Very short and Short Query in NTCIR-2*. *Very short* query is set of the words extracted from a title part in each topic. *Short* query is also set of the words appeared in a description part in each topic.

In order to evaluate effectiveness of the search in our index structure, we applied two query processing to the query terms, *Stopward removal and Term padding*. On *Stopword removal*, we removed the words in the stoplist on SMART test collection from query. The other processing ,*Term padding*, is a little bit special processing for the indexes using *suffix array* family. The indexes were built not on terms, but all substrings in document collection. Some terms might match quite different terms as those subtrings, such as "Some" including "me". Previous work studied this problem, and provided the way to handle it [4]. They experimented three ways ,*prefix, suffix and space*, to solve substring problem using additional white space. *prefix* and *suffix* add single white space at the beginning and end ,respectively, of each query terms. *space* also add single white space both begging and end of each query terms. It marked the highest effectiveness on their experiment. Thus, we used the *space* as term padding to queries.

Query ,for example, is " High-speed ", " transfer " and " TCP " from title on topic "High-speed transfer in TCP".

[*1] http://research.nii.ac.jp/ntcir/index-en.html

Table 1 The number of terms and average number of terms in topic on *E-E task*

Query	# of all the terms	Average # of terms on topic
Title(*Very short*)	131	3
Description(*Short*)	523	11

4.3 Evaluation setting

We use BM-25 similarity metrics(Equality 3) on scoring documents.

$$S(p,d_i) = \sum_{p \in Q} IDF_{part} \cdot \frac{(k_1+1)\cdot tf(p,d_i)}{k_1((1-b)+b\times(L(d_i)/L_{avg}))+tf(p,d_i)} \tag{3}$$

$$IDF_{part} = log\left(\frac{N-df(p)+0.5}{df(p)+0.5}\right)$$

Where parameters $k_1 = 1.2$ $b = 0.5$ are used in the previous work [3], N is the number of documents in the collection, $L(d_i)$ *and* L_{avg} are the length of *i-th* document, and average length of all documents in the collection respectively, $tf(p,d_i)$ is the number of occurrence of pattern p in document d_i, $df(p)$ is the number of individual documents containing pattern p.

We compared our methods(**TA**, **TC**), Culpepper's method(**CUL**) and Exhaustive method(**EXH**) for evaluating effectiveness of search. **TA** cannot handle *df* but *kDF* in scoring, **TC** use *term count* too. **CUL** is not able to use any kind of weighting from terms, because of their pruning traversal. **EXH** is modified method of Culpepper's method to collect every documents within the range on D. Thus, while the efficiency is spoiled, this method can reproduce *df*.

Here, we are able to estimate the importance of each query term with *range*(subsection 3.2). before starting *document array* traverse. Therefore, there are some room to consider the which terms to use. We tried to two manners to choose query terms: balance and small range. First manner is to gather candidate documents on same level for each query terms. For example, to gather three relevant documents, we extract one document from each query terms. Second manner is to prioritize query terms with small *range* high, which is *term count*, and choose to collect candidates along with the terms preferably. In this manner, for example, we gather 3 candidates on one highest query terms in the said setting. The combination of methods and setting are in Table 2.

Table 2 Features of methods: *df* value and pruning

method	*df(p)* in $S(p,d_i)$	pruning	prioritizing terms
TA	approximate *df*	pruning	balance
TA-m	approximate *df*	pruning	small range
TC	*term count*	pruning	small range
CUL	constant	pruning	balance
CUL-m	constant	pruning	small range
EXH	*df*	not pruning	balance

4.4 Evaluating efficiency

We measured the time to process all of query terms in title of *E-collection*, breaking down our method to parameter *k'*, which is the rate($k' \times 10\%$) to the range *ep - sp*. Given a query term with the range $ep - sp = 123$,for example, our method with $k' = 1$ gathers 12 candidate documents relative to the terms. The number of the terms was 93, which return at least one document.

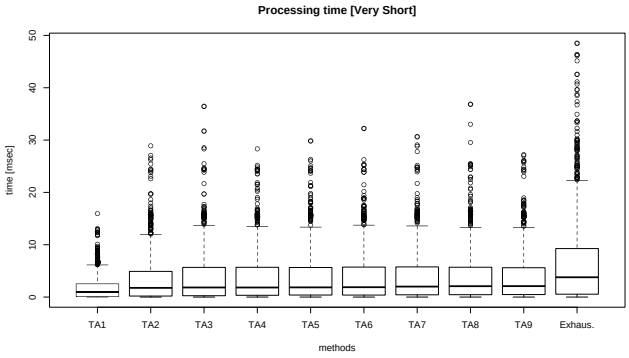

Fig. 3 Processing time for our method and exhaustive method on 93 *Very short* query terms on *E-collection*

Figure 3 shows the processing time of ten run on the terms per methods. Table 3 also shows average of the time. Our method on every parameter is superior to **EXH**, since the pruning increase the efficiency.

Our method with $k' = 1$ was the most efficient among all of the methods. while the time fluctuate widely on its *document frequency*, once the parameter is over 2, the average time were not quite different from on the other(roughly *3.5 msec*). The average time are about a half of the time of **EXH**. Main reason of unchange is that the number of candidate documents is enough large to cover the balance tree structure of *document array*.

4.5 Evaluating effectiveness

We evaluated the effectiveness of search about our methods:**TA** , **TA-m** and **TC** measured by mean average precision(MAP) on all the topic of *E-E task* using both *Very short* and *Short* queries. The result is shown in tables 4 and 5. Since parameter *k'* influence on the coverage of tree structure, the MAP in tables vary with the parameter and accuracy of approximate value. However, all variation of our methods overwhelmed **CUL** on MAP using both queries. Though the differences of MAP between ours with best parameter and **EXH** was tight, our values were lower than exhaustive method.

On the result of *Very short* (Table 4), TA on $k' = 8$ *and* 9 showed statistical significance to Cul on $k' = 9$, increasing its MAP over parameter *k'* gradually. The other variations: TA-m and TC, on the other hand, showed the significance early on the parameter from three to nine. This is because the manner could collect relevant documents. The MAP of TC is slightly lower than TA-m, it indicated that approximate value *kdf* is more appropriate than *tc*. This trend of effectiveness result was similar to the result on *Short* query(Table 5). However, the difference on the number of query terms boosted MAP up to approximate 0.15 at the top of each methods. At the last of effectivenss evaluation, we discuss the accuracy of our approximate value itself in the section A.2, overall the values were super highly correlated with *document frequency*.

4.6 Evaluating efficiency and effectiveness trade-off

At the end of our experiment, we investigated trade-offs between efficiency and effectiveness of search on our method com-

Table 3 Average Processing time for our method and exhaustive method on 93 *Very short* query terms on *E-collection*

	TA & parameter k'									Exh.
k'	1	2	3	4	5	6	7	8	9	
Avg.time	1.861	3.572	3.571	3.780	3.861	3.837	3.922	3.993	3.911	7.02

Table 4 Effectiveness results for **TA**, **CUL** and **EXH** on *Very Short* queries measured by mean average precision. "*" and "**" indicate statistical significance ,based on paired t-test, at $p < 0.005$ *and* $p < 0.001$ levels respectively. The significance were tested relative to Cul with parameter $k' = 9$ for TA and Cul-m with parameter $k' = 9$ for TA-m and TC, on which the method achieved the best mark.

	TA & parameter k'								
k'	1	2	3	4	5	6	7	8	9
MAP	0.0248	0.0404	0.0657	0.0787	0.0848	0.0862	0.0879	0.0955*	0.0958*

	Cul. & parameter k'									Exhaus.
k'	1	2	3	4	5	6	7	8	9	
MAP	0.0211	0.0350	0.0529	0.0618	0.0666	0.0668	0.0676	0.0713	0.0713	0.1138

	TA-m & parameter k'								
k'	1	2	3	4	5	6	7	8	9
MAP	0.0574	0.0864	0.0914	0.0965	0.1026*	0.1041*	0.1040*	0.1048*	0.1055*

	TC & parameter k'								
k'	1	2	3	4	5	6	7	8	9
MAP	0.0594	0.0838	0.0922	0.0982	0.1006*	0.1019*	0.1019*	0.1031*	0.1039*

	Cul-m & parameter k'									Exhaus.
k'	1	2	3	4	5	6	7	8	9	
MAP	0.0310	0.0579	0.0632	0.0680	0.0742	0.0759	0.0757	0.0765	0.0772	0.1139

Table 5 Effectiveness results for **TA**, **CUL** and **EXH** on *Short* queries measured by mean average precision. "*" and "**" indicate statistical significance ,based on paired t-test, at $p < 0.005$ *and* $p < 0.001$ levels respectively. The significance were tested relative to Cul with parameter $k' = 8$ for TA and Cul-m with parameter $k' = 1$ for TA-m an TC, on which the method achieved the best mark.

	TA & parameter k'								
k'	1	2	3	4	5	6	7	8	9
MAP	0.0368	0.0752	0.0991	0.1201*	0.1341**	0.1392**	0.1485**	0.149**	0.1487**

	Cul. & parameter k'									Exhaus.
k'	1	2	3	4	5	6	7	8	9	
MAP	0.0123	0.0382	0.0497	0.0581	0.0724	0.0742	0.0772	0.0776	0.0775	0.1540

	TA-m & parameter k'								
k'	1	2	3	4	5	6	7	8	9
MAP	0.1482*	0.1508**	0.1532**	0.1532**	0.1532**	0.1532**	0.1532**	0.1532**	0.1532**

	TC & parameter k'								
k'	1	2	3	4	5	6	7	8	9
MAP	0.1453*	0.1464**	0.1471**	0.1471**	0.1471**	0.1471**	0.1471**	0.1471**	0.1471**

	Cul-m & parameter k'									Exhaus.
k'	1	2	3	4	5	6	7	8	9	
MAP	0.0804	0.0758	0.0777	0.0777	0.0777	0.0777	0.0777	0.0777	0.0777	0.1572

pared to other methods. Figures 4.6, 4.6 and 4 shows the trade-offs, and these are basically re-plot the above results. The figures showed that our methods were the best on the trade-off. Our method was higher MAP than **CUL** and could reduce processing time compared to **EXH**.

5. Conclusion

We presented the methods approximating *document frequency* on the indexes derived from *suffix array* family without any auxiliary structures. The ideas are making use of the topology of

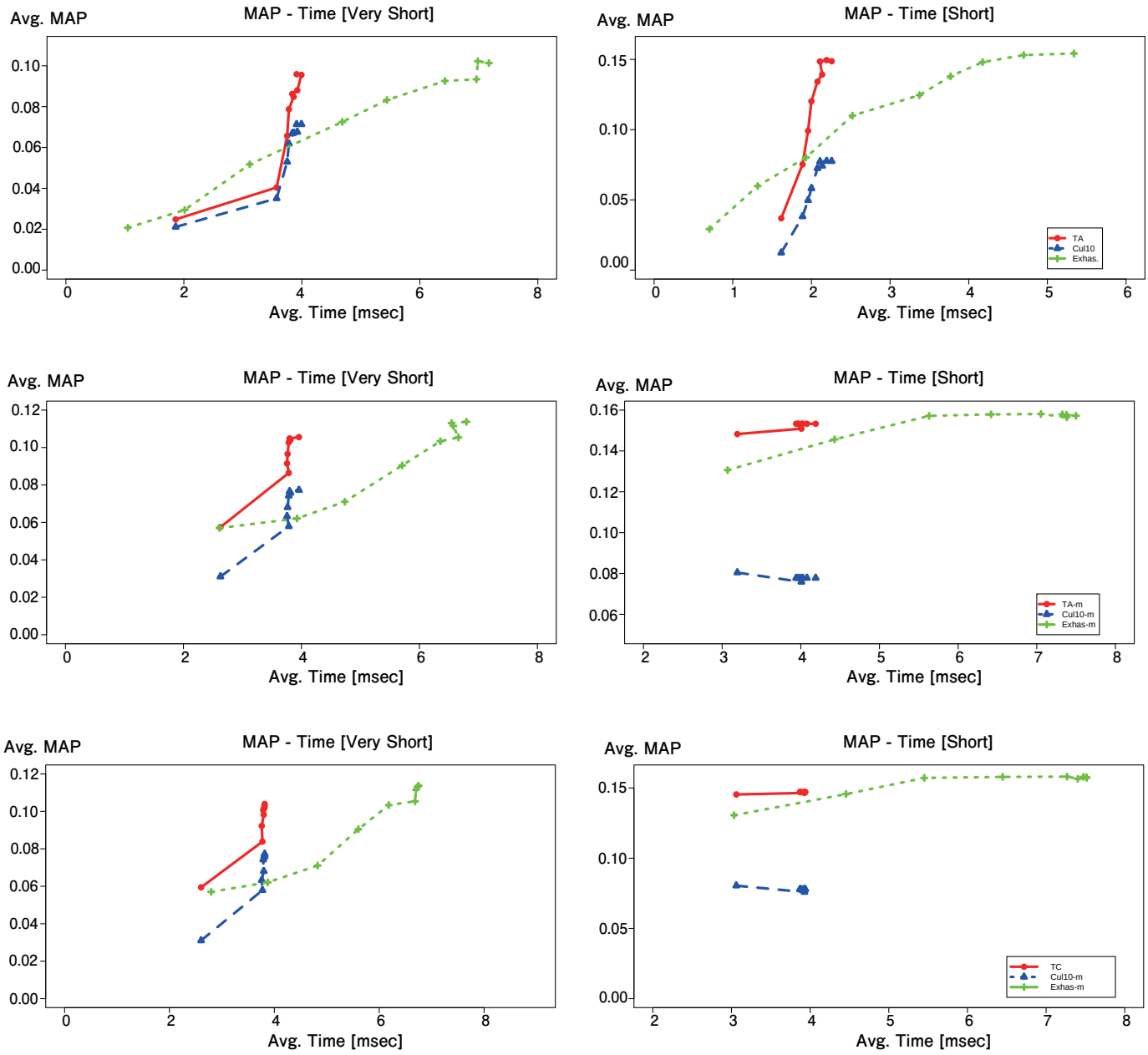

Fig. 4 Average MAP is average of MAP over all the topics, average time is average processing time to process all terms in *Very short* and *Short* queries

wavelet tree on traversing the index and applying *term count* for *df* on our observation; *range* in the index equals *term count*. We also investigated the efficiency and effectiveness of search on NTCIR-2 test collection in details, and showed our methods and approximate calculation work well in practical setting. This work is an important mile stone on this research fields, because our methods could grasp term weight(approximate *df*) in the index and on time to search. It makes possible to develop methods that eliminate inrelevant part of indexed documents on traversing time. This term weight are actually for arbitrary indexed substrings. This fact can lead this index structure to the framework on other retrieval tasks where we cannot be define terms in advance.

It is still required the approximate methods and novel ranking metrics specifically on this indexes framework for practical index framework. As to our methods, there are some room to optimize the methods by applying heuristics studied in inverted index. We will investigate the way to handle substrings or phrases in search for further improvement of effectiveness for future work.

References

[1] Anh, V. N. and Moffat, A.: Pruned query evaluation using pre-computed impacts, *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, ACM, pp. 372–379 (2006).

[2] Culpepper, J. S., Navarro, G., Puglisi, S. J. and Turpin, A.: Top-k ranked document search in general text databases, *Proceedings of the 18th Annual European Symposium on Algorithms (ESA 2010)*, ESA'10, Berlin, Heidelberg, Springer-Verlag, pp. 194–205 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1882123.1882145⟩ (2010).

[3] Culpepper, J. S., Petri, M. and Scholer, F.: Efficient in-memory top-k document retrieval, *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, New York, NY, USA, ACM, pp. 225–234 (online), DOI: 10.1145/2348283.2348317 (2012).

[4] Culpepper, J. S., Yasukawa, M. and Scholer, F.: Language Independent Ranked Retrieval with NeWT, *Proceedings of the 16th Australian Document Computing Symposium (ADCS 2011)*, Canberra, pp. 18–25 (online), available from ⟨http://goanna.cs.rmit.edu.au/ e76763/publications/cys11-adcs.pdf⟩ (2011).

[5] Fontoura, M., Josifovski, V., Liu, J., Venkatesan, S., Zhu, X. and Zien, J. Y.: Evaluation Strategies for Top-k Queries over Memory-Resident Inverted Indexes, *Proceedings of 37th International Conference on Very Large Data Bases (VLDB 2011)*, Vol. 4, No. 12, pp. 1213–1224 (2011).

[6] Gagie, T., Puglisi, S. and Turpin, A.: Range Quantile Queries: Another Virtue of Wavelet Trees, *Proceedings of the 16th International Symposium on String Processing and Information Retrieval (SPIRE 2009)* (Karlgren, J., Tarhio, J. and Hyyrö, H., eds.), Lecture Notes in Computer Science, Vol. 5721, Springer Berlin Heidelberg, pp. 1–6 (online), DOI: 10.1007/978-3-642-03784-9_1 (2009).

[7] Hon, W.-K., Shah, R. and Vitter, J.: Space-Efficient Framework for Top-k String Retrieval Problems, *50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, FOCS '09, pp. 713–722 (online), DOI: 10.1109/FOCS.2009.19 (2009).

[8] Klein, M. and Nelson, M. L.: Approximating Document Frequency with Term Count Values, *Computing Research Repository (CoRR)*, Vol. abs/0807.3755 (2008).

[9] Klein, M. and Nelson, M. L.: A Comparison of Techniques for Estimating IDF Values to Generate Lexical Signatures for the Web, *Proceedings of 10th ACM International Workshop on Web Information and Data Management*, WIDM 2008 (2008).

[10] Navarro, G. and Nekrich, Y.: Top-k document retrieval in optimal time and linear space, *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, SODA '12, SIAM, pp. 1066–1077 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2095116.2095200⟩ (2012).

[11] Navarro, G. and Valenzuela, D.: Space-Efficient Top-k Document Retrieval, *Proceedings of 11th International Symposium on Experimental Algorithms (SEA 2012)* (Klasing, R., ed.), Lecture Notes in Computer Science, Vol. 7276, Springer Berlin Heidelberg, pp. 307–319 (online), DOI: 10.1007/978-3-642-30850-5_27 (2012).

Appendix

A.1 Basic concepts

In this section, we refer to the algorithms and data structure consisting our index structure. We pick up fundamental operations: *Rank and Select* in compact data structure, *Wavelet tree* and *FM-index* as key features of our system to explain.

A.1.1 Rank/Select on binary sequence

Let us define two operation on binary sequence B[0,n-1].

$rank_b(B, i)$ returns the number of occurrence of bit b in B[0,i]
$select_b$(B, j) returns the position of jth appearance of bit b in B

Two simple operations on B are essential for compact data structures. The efficiency of the data structures are depend on these operations, because the operations are unit functions for calculating B.

A.1.2 Rank/Select on general sequence

Let extend $rank/select$ operation to over general sequence S[0, n-1], alphabet size $|\Sigma| > 2$. These operations can be supported by wavelet trees. Basic operations on wavelet trees as follows:

$rank_s(S, c, i)$ returns the number of occurrence of symbol c in S[0, i]
$select_s(S, c, j)$ returns the position of jth occurrence of symbol c in S[1, i]
$access(S, i)$ returns the symbol S[i]

A wavelet tree over over alphabet Σ is a binary balanced tree with σ leaves. The root node store binary B_{root} defined as follows: if $S[i] \leq \sigma/2$(that is, $S[i]$ is descendant on alphabetical order) then B_{root}[i] = 0, else B_{root}[i] = 1. Subtexts S_0 is formed by concatenating symbols $S[i]$ where $B_{root}[i] = 0$, and S_1 is also formed by same manner concatenating $S[i]$ where $B_{root}[i] = 1$. Then, S_0 is allocated to left node and S_1 is allocated to right node. The wavelet tree can be construct above procedures recursively to $\log\sigma$ level. Important properties of the tree are each leaf indicates individual symbol c and the height is $\log\sigma$. $rank_s/select_s$ operations computed by $rank_b/select_b$ operations on each level on from root to leaf node and combining the results. Therefore, $rank_s/select_s$ operations can computed in $O(\log\sigma)$. $access$ operation ,returning any symbol c at $S[i]$, can be supported in $O(\log\sigma)$.

A.1.3 FM-Index

Self-index is a data structure that index a text T, and supports to the following operations:

$locate(p)$ returns the positions where pattern p appears in T
$count(p)$ returns the number of occurrence of pattern p in T

Fig. A·1 Approximate values and *document frequency*

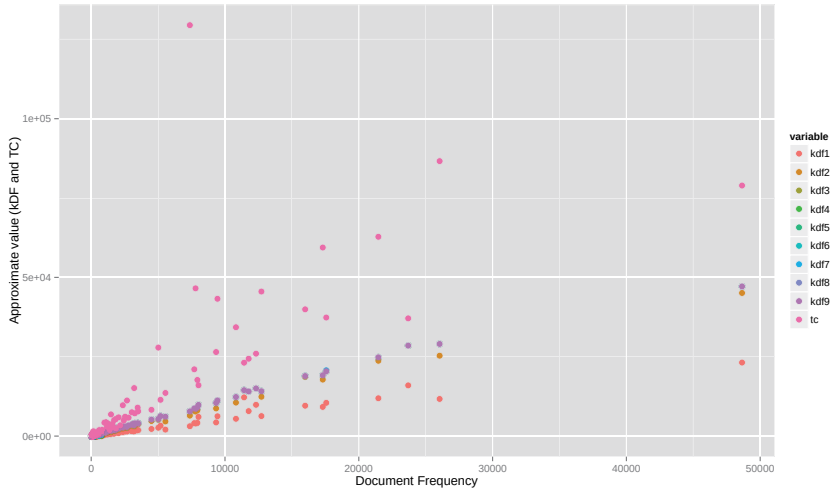

Table A·1 Correlation coefficient

	kdf									tc
k'	1	2	3	4	5	6	7	8	9	
R	0.9915	0.9961	0.9969	0.9969	0.9982	0.9986	0.9992	0.9995	0.9996	0.9804

$extract(sp, ep)$ returns the substring $T[sp, ep]$

A suffix array $SA[0, n-1]$ is in the class of self-index, built on top of T. SA is a permutation of all suffixes positions $0 \,..\, n-1$ in T, where the suffixes are in lexicographical order. Given pattern $p[0, m-1]$ occurring in T, $SA[sp, ep]$(range) is determined by two *locate* operations on SA. The suffixes sharing p as the prefix are gathered in this range $SA[sp, ep]$, because of lexicographical order within SA.

We can determine a range *sp and ep* along with p on T^{BWT} transformed T by Burrows-Wheeler Transform with some functions, which is *FM-Index*. T^{BWT} is just a permutation of T. It is generated by sorting n cyclic shifts of T in lexicographical order, then, picking the last column of $n * n$ matrix up; corresponding T^{BWT}. The range *sp and ep* on *FM-Index* can be computed with equation below;

$$sp_i = C[p[i]] + rank_s(T^{BWT}, p[i], sp_{i+1} - 1) \quad \text{(A.1)}$$

$$ep_i = C[p[i]] + rank_s(T^{BWT}, p[i], ep_{i+1}) - 1 \quad \text{(A.2)}$$

Where $C[]$ a is look up table storing the occurrence of symbols smaller than symbol $p[i]$ in T. First, starting with the last symbol of $p[m-1]$ and $sp_{m-1} = 0, ep_{m-1} = m-1$, we update sp_i *and* ep_i using the range sp_{i+1} *and* ep_{i+1} corresponding to the previous symbols $p[i+1, m-1]$.

A.2 Evaluation of approximate document frequency

We evaluated accuracy of our approximate values: *kDF and term count* to measure correlation coefficient between the values and *(real) document frequency*. We used all terms in *Very short* query and on our estimating method(*kDF*) with parameter *k'* and *term count* for this evaluation. Table A·1 indicate that all of approximate values remarked high correlation with *(read) df*. Those values are over 0.98. Though, *term count* marked high correlation with *df* enough, the value is the lowest of all. Figure A·1 displayed this result; *term count* was more scattered than *kDF* values.